The untyped computational λ-calculus and its intersection type discipline

The untyped computational λ-calculus and its intersection type discipline

Ugo de' Liguoro[a], Riccardo Treglia[a]

[a] *Università di Torino, c.so Svizzera 185, 10149 Turin, Italy*
ugo.deliguoro@unito.it, riccardo.treglia@unito.it

Abstract

We study a Curry style type assignment system for untyped λ-calculi with effects, based on Moggi's monadic approach. Moving from the abstract definition of monads, we introduce a version of the call-by-value computational λ-calculus based on Wadler's variant, without *let*, and with unit and bind operators. We define a notion of reduction for the calculus and prove it confluent.

We then introduce an intersection type system inspired by Barendregt, Coppo and Dezani system for ordinary untyped λ-calculus, establishing type invariance under conversion.

Finally, we introduce a notion of convergence, which is precisely related to reduction, and characterize convergent terms via their types.

Keywords: Monads, Reduction, Type assignment systems

1. Introduction

The computational λ-calculus was introduced by Moggi in [1, 2] as a meta-language to describe non functional effects in programming languages via an incremental approach. The basic idea is to distinguish among values of some type D and computations over such values, the latter having type TD. Semantically T is a monad, endowing D with a richer structure such that operations over computations can be seen as algebras of T.

The monadic approach is not only useful when building compilers modularly with respect to various kinds of effects [2], to interpret languages with effects like control operators via a CPS translation [3], or to write effectful programs in a purely functional language such as Haskell [4], but also to reason

about such programs. In this respect, typed computational lambda-calculus has been related to static program analysis and type and effect systems [5, 6], PER based relational semantics [7], and more recently co-inductive methods for reasoning about effectful programs have been investigated [8].

Aim of this work is to investigate the monadic approach to effectful functional languages in the untyped case. This is motivated by the fact that similar, if not more elusive questions arise for effectful untyped languages as well as for typed ones. The tools we use to establish program equivalence are classic theory of reduction and intersection type assignment, which we exploit for defining the logical semantics of programs.

It might appear nonsense to speak of monads w.r.t. an untyped calculus, as the monad T interprets a type constructor both in Moggi's and in Wadler's formulation of the computational λ-calculus [2, 4]. However, much as the untyped λ-calculus can be seen as a calculus with a single type, as formerly observed by Scott [9], the untyped computational λ-calculus, here dubbed λ_c^u, has two types: the type of values D and the type of computations TD. Semantically this involves the existence of a call-by-value reflexive object in the categorical model [1].

Reduction and operational semantics of the computational λ-calculus have been studied in the context of call-by-need calculi, e.g. in [10, 11], and confluence of reduction of Moggi's untyped λ_c has been established recently in [12], depending on the strong normalization of the calculus. However the calculus we consider here is much simpler, strictly speaking not a sub-calculus of any of its ancestors, as it doesn't have neither the *let*, nor functional application as primitive constructs. The syntax is derived from Wadler's formalization of monads, in terms of unit and bind operations, written `return` and `>>=` in Haskell idiom. Reduction is then defined simply by orienting equations of the three monad laws, and this suffice to obtain a perfectly meaningful calculus with confluent reduction relation.

The main concern of our paper is the definition of an intersection type assignment system for the untyped computational λ-calculus. Intersection types are an extension of Curry's simple types introduced in the 80's, such that relevant classes of λ-terms are characterized by means of their types: see [13] Part III, and the references there. The motivation for investigating intersection types is that, when including a universal type, usually denoted by ω, that can be assigned to any term, types are invariant under term conversion, instead of just reduction; by this property the term meaning, namely its functional behaviour, is fully characterized by the set of its types.

So, having such a system for the computational λ-calculus, opens the way to study by well established mathematical tools the case of (untyped) λ-calculi with effects.

To substantiate our claims, we consider a straightforward adaptation of Abramsky's idea of convergence for the lazy λ-calculus, as the only observable property of terms [14, 15]. We then relate convergence to reducibility to the trivial computation of a value; eventually we show that convergent terms are exactly those that have a type different than the universal type of computations.

Summary. The paper is organized as follows: in section 2 we recall Moggi's definition of a λ_c-model, in the setting of concrete categories. The syntax and semantics of our calculus are presented in section 3. Then, in section 4, we study reduction and prove its confluence in section 5. In section 6 we introduce the type assignment system, and in section 7 we establish the subject reduction and expansion theorems. The convergence predicate is defined in section 8; it is related to the reduction relation and the characterization theorem is established. Finally in section 9 we discuss related works and propose some further developments. Then we conclude.

This is a revised version of the unpublished paper [16]. With respect to the draft, the present paper doesn't include the modelling of types, nor soundness and completeness of the type system w.r.t. the type interpretation, to limit space and allow a lighter reading. However, since the subject expansion property was established by means of semantic tools in the draft, we have included here a syntactical proof of the same result.

2. Concrete models of the computational λ-calculus

The computational λ-calculus, denoted λ_c, has been introduced in the seminal works [1, 2]. It is a typed calculus derived from the categorical construction of a monad (T, η, μ) (see [17] chap. VI) over a cartesian category $\mathcal{C}$, equipped with some more structure to model Kleisli exponents, which represent internally the morphisms in $\mathcal{C}_T(A, B) = \mathcal{C}(A, TB)$, where $\mathcal{C}_T$ is the Kleisli category of the monad. For the precise definition see [1], Defs. 3.2 and 3.5, or [2] Defs. 3.2 and 3.9; see also Def. 2.2 and Props. 2.3 and 2.4 below.

As said before, in Moggi's construction $\mathcal{C}$ is cartesian. When looking at Wadler's type-theoretic definition of monads [18, 4], that is at the basis of

their successful implementation in Haskell language, a natural interpretation of the calculus is into a cartesian closed category (ccc), such that two families of combinators, or a pair of polymorphic operators called the "unit" and the "bind", exist satisfying the *monad laws*, namely (the syntactic counterpart of) the three equations in Def. 2.1 below (see also Prop. 3.4). This is more directly expressed by defining the interpretation of Wadler's version of the λ_c-calculus into a (locally small) subcategory of **Set** which is a ccc: here $\mathcal{C}$ will be called a *concrete ccc*. Examples are the category **Dom** of Scott domains with continuous functions, and its subcategory **Alg** of algebraic lattices with a countable basis.

Definition 2.1. *Let $\mathcal{C}$ be a concrete ccc. A* functional computational monad*, henceforth* functional monad *over $\mathcal{C}$ is a triple $(T, unit, \star)$ where T is a map over the objects of $\mathcal{C}$, and unit and $\star$ are families of morphisms*

$$unit_A : A \to TA \qquad\qquad \star_{A,B} : TA \times (TB)^A \to TB$$

such that, writing $\star_{A,B}$ as an infix operator and omitting subscripts:

$$\begin{array}{lrcl} \textit{Left unit}: & unit\, a \star f & = & f\, a \\ \textit{Right unit}: & m \star unit & = & m \\ \textit{Assoc}: & (m \star f) \star g & = & m \star \lambda\!\!\lambda d.(f\, d \star g) \end{array}$$

where $\lambda\!\!\lambda$ is functional abstraction in the metalanguage: $\lambda\!\!\lambda d.(f\, d \star g)$ means $d \mapsto f\, d \star g$.

This definition is the semantic counterpart of the type theoretic one in [4], but for the $\star$ which is curried in [18] and Wadler's subsequent papers, so that it has type $TA \to (A \to TB) \to TB$; we adopt here the uncurried form to avoid the cumbersome double exponent.

Leaving aside the discussion about different ways to define a λ_c-model over a ccc in general, for which the interested reader might consult [19] and the literature cited there, we limit ourself to show that a functional monad is indeed a strong monad and a λ_c-model in the sense of Moggi. Clearly Def. 2.1 is quite close to the notion of a *Kleisli triple* $(T, \eta, \_^*)$, which is an equivalent definition of a monad.

Definition 2.2. *A* Kleisli triple *over a category $\mathcal{C}$ is a structure $(T, \eta, \_^*)$ where $T : Obj_{\mathcal{C}} \to Obj_{\mathcal{C}}$ is a map over the objects of $\mathcal{C}$, and there are a family of morphisms $\eta_A : A \to TA$ of $\mathcal{C}$ and a map $\_^*$, we refer to as the*

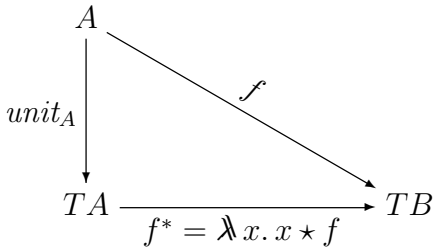
extension map *of* T*, sending any morphism* $f : A \to TB$ *in* $\mathcal{C}$ *to a morphism* $f^* : TA \to TB$ *of the same category, such that:*

$$f^* \circ \eta_A = f, \qquad \eta_A^* = \mathrm{id}_{TA}, \qquad f^* \circ g^* = (f^* \circ g)^* \tag{1}$$

where $g : C \to TA$*.*

Proposition 2.3. *A functional monad* $(T, unit, \star)$ *over a concrete ccc* $\mathcal{C}$ *induces a Kleisli triple over* $\mathcal{C}$*.*

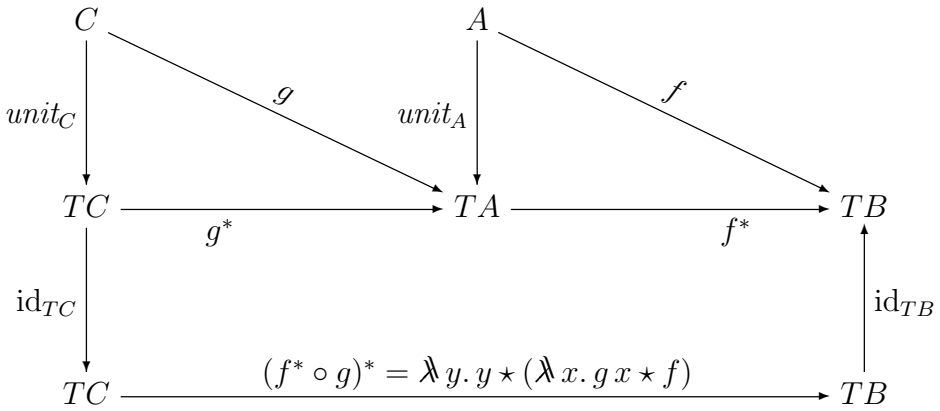
Proof. Take $\eta_A = unit_A$ and $f^* = \lambda\!\!\lambda\, x.x \star f : TA \to TB$ for $f : A \to TB$. By the equation (*Left unit*) the following diagram commutes:



By definition we have that $\eta_A^* = \lambda\!\!\lambda\, x.x \star unit_A$; therefore by (*Right unit*), for all $a \in TA$ we have:

$$\eta_A^*\, a = (\lambda\!\!\lambda\, x.x \star unit_A)\, a = a \star unit_A = a$$

Finally by (*Assoc*) the following diagram commutes:



namely for all $c \in TC$:

$$(f^* \circ g)^*\, c = c \star (\lambda\!\!\lambda\, x.\, g\, x \star f) = (c \star g) \star f$$

□

The above proof relies on the inter-definability of extension and bind by the equation $f^*a = a \star f$. On the other hand by looking closely to this equation, we see that what we have constructed is the morphism:

$$\_^* = \lambda\!\!\lambda\, f\, x.\, x \star f : TB^A \to TB^{TA} \tag{2}$$

internalizing the Kleisli map. This is the essential step in the construction of what is called a $\mathcal{C}$-monad in [19], §5.

Since then we have not completely exploited the fact that $\mathcal{C}$ is a ccc. As such it has all finite products, so that for any A, B the morphism

$$t_{A,B} : A \times TB \to T(A \times B)$$

is definable in terms of $\star$, pairing and projections as

$$t_{A,B} = \lambda\!\!\lambda\, x.\, (\pi_2\, x) \star \lambda\!\!\lambda\, y.\, unit\, (\pi_1\, x, y) \tag{3}$$

which is such that:

$$t\, (a, m) = m \star \lambda\!\!\lambda\, y.unit\, (a, y) \tag{4}$$

Then t is a *tensorial strength* in the sense of [2], Def. 3.2, as stated in the following proposition.

Proposition 2.4. *Given a monad $(T, unit\,, \star)$ the t defined in equation (3) commutes with the natural isomorphisms $r_A : 1 \times A \to A$ and $\alpha_{A,B,C} : (A \times B) \times C \to A \times (B \times C)$:*

i) $t_{1,A} \circ r_{TA} = Tr_A$

ii) $T\alpha_{A,B,C} \circ t_{A\times B,C} = t_{A,B\times C} \circ (\mathrm{id}_A \times t_{B,C}) \circ \alpha_{A,B,TC}$

Moreover:

iii) $t_{A,B} \circ (\mathrm{id}_A \times unit_B) = unit_{A\times B}$

iv) $t_{A,B} \circ (\mathrm{id}_A \times \mu_B) = \mu_{A\times B} \circ Tt_{A,B} \circ t_{A,TB}$

where $\mathrm{id}_A = \lambda\!\!\lambda\, x{:}A.\, x$, $\mu_A = (\mathrm{id}_{TA})^* = \lambda\!\!\lambda\, z.\, z \star \mathrm{id}_{TB}$ *and for any* $f : A \to B$,

$$Tf = (unit_B \circ f)^* = \lambda\!\!\lambda\, z.\, z \star (unit_B \circ f)$$

Therefore $(T, unit\,, \star, t)$ *is a* strong monad.

Proof. By definition unfolding and straightforward calculations. □

3. Untyped λ_c-calculus

Much as the untyped λ-calculus can be seen as a calculus with a single type D such that $D = D \to D$, the *untyped computational λ-calculus*, that we dub λ_c^u, has two types: the type of *values* D and the type of *computations* TD. In [1], §5 the semantics of these types is given by two kinds of reflexive objects in the category of λ_c-models, that as type equations read either as $D = TD \to TD$ in case of call-by-name, or

$$D = D \to TD \tag{5}$$

in case of call-by-value. Since the distinction among values and computations is central in λ_c, which has been conceived as a generalization of Plotkin's call-by-value λ-calculus, we adopt equation (5) in defining the corresponding untyped calculus, which leads to the following definition of the λ_c^u syntax:

Definition 3.1 (Terms of λ_c^u)**.** *The terms of the* untyped computational λ-calculus*, shortly λ_c^u, consist of two sorts of expressions:*

$$\begin{array}{rrcll} \mathit{Val}: & V, W & ::= & x \mid \lambda x.M & \textit{(values)} \\ \mathit{Com}: & M, N & ::= & \mathit{unit}\, V \mid M \star V & \textit{(computations)} \end{array}$$

where x ranges over a denumerable set Var of variables. We set $\mathit{Term} = \mathit{Val} \cup \mathit{Com}$; $FV(V)$ and $FV(M)$ are the sets of free variables occurring in V and M respectively, and are defined in the obvious way. Terms are identified up to clash avoiding renaming of bound variables (α-congruence).

Because of (5) it easily seen that, if we assume that all variables x have type D, then any value term V has type D and any computation term has type TD. Indeed omitting contexts, we have:

$$\begin{array}{ccc} x : D \vdash x : D & & \dfrac{x : D \vdash M : TD}{\lambda x.M : D \to TD = D} \\[2ex] \dfrac{V : D}{\mathit{unit}\, V : TD} & & \dfrac{M : TD \qquad V : D = D \to TD}{M \star V : TD} \end{array} \tag{6}$$

With respect to Moggi's λ_c-syntax, we do not have the *let* construct, which is considered as syntactical sugar for bind and abstraction:

$$\mathit{let}\ x = N\ \mathit{in}\ M \ \equiv\ N \star \lambda x.M$$

Notably we do not have application in the syntax. In fact, while VW might be included, yielding a term of type TD, none among MV, VM and MN have a type in the calculus; indeed these are not well formed terms according to Def. 3.1. However, there is a deeper reason for dropping application from the primitive operators: as a matter of fact the bind operator of a monad is a postfix functional application [18], whose actual definition depends on the monad itself.

We end this part by defining the notion of λ_c^u-model in a concrete ccc. by analogy to that of environment λ-model in [20].

Definition 3.2 (Reflexive T-object and λ_c^u-model)**.** *Let $\mathcal{C}$ be a concrete ccc, and $(T, unit\,, \star)$ a functional monad over $\mathcal{C}$. Then an object $D \in Obj_{\mathcal{C}}$ is* T-reflexive *if there exist the $\mathcal{C}$-morphisms $\Phi : D \to TD^D$ and $\Psi : TD^D \to D$ such that $\Phi \circ \Psi = \mathrm{id}_{TD^D}$.*

*A λ_c^u-*model *in $\mathcal{C}$ is a tuple $\mathcal{M}(D) = (D, T, \Phi, \Psi)$ where T is a monad, D is a reflexive T-object via Φ and Ψ. Then, setting Term-Env$_D$ $= Var \to D$ as the set of* variable environments *ranged over by ρ, we define a pair of maps $[\![\cdot]\!]^D$: Val $\times$ Term-Env$_D$ $\to D$ and $[\![\cdot]\!]^{TD}$: Com $\times$ Term-Env$_D$ $\to D$, such that:*

i) $[\![x]\!]^D_\rho = \rho(x)$

ii) $[\![\lambda x.M]\!]^D_\rho = \Psi(\lambda\!\!\lambda\, d \in D.[\![M]\!]^{TD}_{\rho[x \mapsto d]})$

iii) $[\![unit\, V]\!]^{TD}_\rho = unit\, [\![V]\!]^D_\rho$

iv) $[\![M \star V]\!]^{TD}_\rho = [\![M]\!]^{TD}_\rho \star \Phi([\![V]\!]^D_\rho)$

where $\rho[x \mapsto d](x) = d$ and $\rho[x \mapsto d](y) = \rho(y)$ if $y \not\equiv x$. Finally we say that $\mathcal{M}$ is extensional *if also $\Psi \circ \Phi = \mathrm{id}_D$.*

As in case of environment λ-models, that interpretations $[\![\cdot]\!]^D$ and $[\![\cdot]\!]^{TD}$ are well defined depends on the fact that $\lambda\!\!\lambda\, d \in D.[\![M]\!]^{TD}_{\rho[x \mapsto d]} \in TD^D$, which is easily established by induction over M. In the following we shall write D for a λ_c^u-model $\mathcal{M}(D)$ whenever the context is unambiguous, and call it just a model.

By $M[V/x]$ and $W[V/x]$ we denote the capture avoiding substitution of x by V in M and W respectively. This means that x is not bound in M, W and that V is free for x in M, namely $FV(V) \cap BV(M) = \emptyset$, and similarly for W. Since we have a denumerable set of variables and identify α-congruent terms, such conditions can always be satisfied.

Lemma 3.3. *Let D be a model. Then for all $V, W \in Val$ and $M \in Com$, and for all $\rho \in Term\text{-}Env_D$:*

$$[\![W[V/x]]\!]^D_\rho = [\![W]\!]^D_{\rho[x \mapsto [\![V]\!]^D_\rho]} \quad \textit{and} \quad [\![M[V/x]]\!]^{TD}_\rho = [\![M]\!]^{TD}_{\rho[x \mapsto [\![V]\!]^D_\rho]}$$

Proof. By an easy induction over W and M. □

For any model D and $M, N \in Com$, we write $D \models M = N$ if for all $\rho \in Term\text{-}Env_D$ it holds $[\![M]\!]^{TD}_\rho = [\![N]\!]^{TD}_\rho$. Then we write $\models M = N$ if $D \models M = N$ for any model D.

Proposition 3.4 (Monad laws)**.** *For all $V \in Val$ and $M, N, L \in Com$ it holds that:*

i) $\models unit\, V \star (\lambda x.M) = M[V/x]$

ii) $\models M \star \lambda x.unit\, x = M$

iii) $\models (L \star \lambda x.M) \star \lambda y.N = L \star \lambda x.(M \star \lambda y.N)$ *where* $x \notin FV(N)$.

Proof. By definition unfolding and easy calculations. E.g. for arbitrary D and $\rho \in Term\text{-}Env_D$:

$$\begin{aligned}
[\![unit\, V \star (\lambda x.M)]\!]^{TD}_\rho &= unit\, [\![V]\!]^D_\rho \star \Phi(\Psi(\lambda\!\!\lambda\, d \in D.[\![M]\!]^{TD}_{\rho[x \mapsto d]})) \\
&= unit\, [\![V]\!]^D_\rho \star \lambda\!\!\lambda\, d \in D.[\![M]\!]^{TD}_{\rho[x \mapsto d]} \\
&= (\lambda\!\!\lambda\, d \in D.[\![M]\!]^{TD}_{\rho[x \mapsto d]})[\![V]\!]^D_\rho \\
&= [\![M]\!]^{TD}_{\rho[x \mapsto [\![V]\!]^D_\rho]} = [\![M[V/x]]\!]^{TD}_\rho
\end{aligned}$$

using Def. 3.2, equations $\Phi \circ \Psi = \mathrm{id}_{TD^D}$, *Left unit* and Lem. 3.3. □

4. Reduction

Since we are considering a calculus where T is a generic monad and monad laws are true in any model, we take the latter as abstractly defining the meaning of *unit* and $\star$, and obtain the following reduction relation by orienting equations in Prop. 3.4.

Following [21] §3.1, we say that a binary relation $R \subseteq Term \times Term$ is a *notion of reduction*. If R_1, R_2 are notions of reductions we abbreviate $R_1 R_2 = R_1 \cup R_2$; then we denote by $\longrightarrow_R$ the *compatible closure* of R, namely the least relation including R which is closed under arbitrary contexts.

Definition 4.1 (Reduction)**.** *The relation* $\lambda\boldsymbol{C} = \beta_c \cup id \cup comp$ *is union of the following binary relations over Com:*

$$\begin{array}{rcl} \beta_c & = & \{\langle unit\, V \star (\lambda x.M), M[V/x]\rangle \mid V \in Val, M \in Com\} \\ id & = & \{\langle M \star \lambda x.unit\, x, M\rangle \mid M \in Com\} \\ comp & = & \{\langle (L \star \lambda x.M) \star \lambda y.N, L \star \lambda x.(M \star \lambda y.N)\rangle \mid L, M, N \in Com, x \notin FV(N)\} \end{array}$$

where $M[V/x]$ *denotes the capture avoiding substitution of* V *for* x *in* M*.*

Finally $\longrightarrow = \longrightarrow_{\lambda\boldsymbol{C}}$ *is the compatible closure of* $\lambda\boldsymbol{C}$*.*

A more readable writing of the definition of $\longrightarrow$ in Def. 4.1 is:

$$\begin{array}{rrcll} \beta_c) & unit\, V \star (\lambda x.M) & \longrightarrow & M[V/x] & \\ id) & M \star \lambda x.unit\, x & \longrightarrow & M & \\ comp) & (L \star \lambda x.M) \star \lambda y.N & \longrightarrow & L \star \lambda x.(M \star \lambda y.N) & \text{for } x \notin FV(N) \end{array}$$

Rule β_c is reminiscent of the left unit law in [4]; we call it β_c because it performs call-by-value β-contraction in λ_c^u. In fact, by reading $\star$ as postfix functional application and merging V into its trivial computation $unit\, V$, β_c is the same as β_v in [22]:

$$(\lambda x.M)V \equiv unit\, V \star (\lambda x.M) \longrightarrow M[V/x] \tag{7}$$

The compatible closure of the relation $\beta_c \cup id \cup comp$ is explicitly defined by means of the typed contexts:

$$\begin{array}{lrcl} \textit{Value contexts:} & \mathcal{V} & ::= & \langle\cdot_D\rangle \mid \lambda x.\mathcal{C} \\ \textit{Computation contexts:} & \mathcal{C} & ::= & \langle\cdot_{TD}\rangle \mid unit\, \mathcal{V} \mid \mathcal{C} \star V \mid M \star \mathcal{V} \end{array}$$

Contexts have just one hole, which is either $\langle\cdot_D\rangle$ or $\langle\cdot_{TD}\rangle$. These are sorted in the sense that they can be replaced only by value and computation terms, respectively. Denoting by $\mathcal{V}\langle P\rangle$ and $\mathcal{C}\langle P\rangle$ the replacements of hole $\langle\cdot_D\rangle$ or $\langle\cdot_{TD}\rangle$ in $\mathcal{V}$ and $\mathcal{C}$ by $P \equiv V$ or $P \equiv M$ (possibly catching free variables in P), we get terms in *Val* and *Com*, respectively. Then compatible closure is expressed by the rule:

$$\frac{M \longrightarrow M'}{\mathcal{C}\langle M\rangle \longrightarrow \mathcal{C}\langle M'\rangle} \tag{8}$$

Using rule (8), the correspondence of rule β_c to β_v can now be illustrated more precisely. First observe that the reduction relation $\longrightarrow$ is only among computations, therefore no computation N will ever reduce to some value V; however, this is represented by a reduction $N \stackrel{*}{\longrightarrow} \mathit{unit}\, V$, where $\mathit{unit}\, V$ is the coercion of the value V into a computation. Moreover, let us assume that $M \stackrel{*}{\longrightarrow} \mathit{unit}\, (\lambda x.M')$; by setting

$$MN \equiv M \star (\lambda z.N \star z) \qquad \text{for} \qquad z \notin FV(N) \tag{9}$$

we have:

$$\begin{array}{lll} MN & \stackrel{*}{\longrightarrow} & \mathit{unit}\, (\lambda x.M') \star (\lambda z.\mathit{unit}\, V \star z) \quad \text{by rule (8)} \\ & \longrightarrow & \mathit{unit}\, V \star (\lambda x.M') \quad \text{by } \beta_c \\ & \longrightarrow & M'[V/x] \quad \text{by } \beta_c \end{array}$$

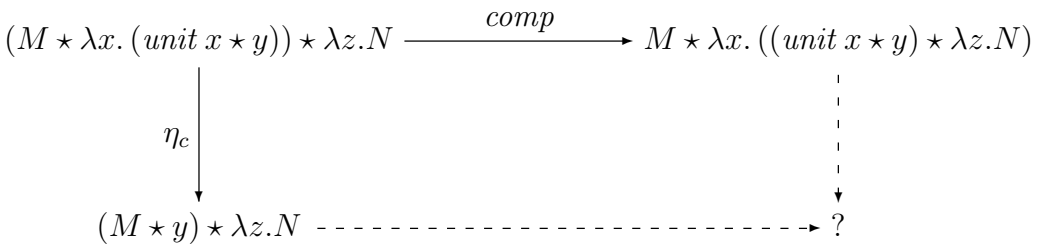
where if $z \notin FV(N)$ then $z \notin FV(V)$ as it can be shown by a routine argument.

We end this section by considering the issue of weak and full extensionality, that have not been treated in Def. 4.1. Weak extensionality, also called ξ-rule of the ordinary λ-calculus, is reduction under abstraction. This is guaranteed by rule (8), but only in the context of computation terms.

Concerning extensionality an analogous of η-rule is:

$$\eta_c) \quad \lambda x.\, (\mathit{unit}\ x \star V) \longrightarrow_{\eta_c} V, \qquad x \notin FV(V) \tag{10}$$

This involves extending reduction from *Com* to the whole *Term*. However the reduction obtained by adding η_c to $\lambda\mathbf{C}$ is not confluent:

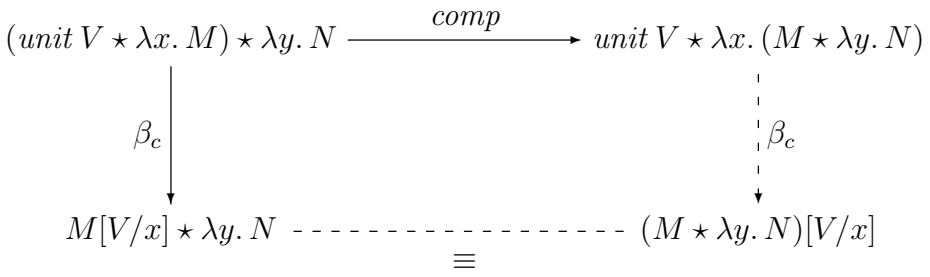


5. Confluence

A fundamental property of reduction in ordinary λ-calculus is confluence, established in the Church-Rosser theorem. In this section we prove confluence

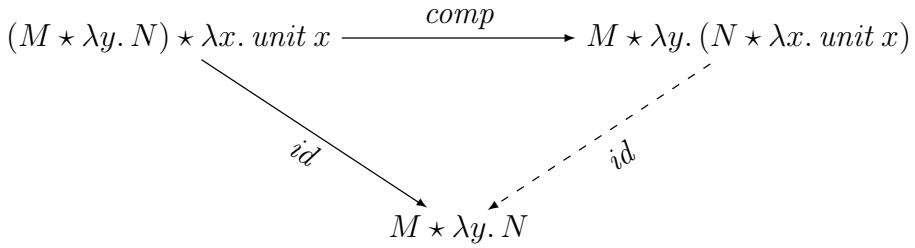
of $\longrightarrow$ for the λ_c^u-calculus. This is a harder task since reduction in λ_c^u has three axioms instead of just the β-rule of the λ-calculus, whose left-hand sides generate a number of critical pairs. Before embarking into the proof, let us see a few examples.

Example 5.1. *In this example we see how outer reduction by comp may overlap with an inner reduction by β_c. Representing the given reductions by solid arrows, we see how to recover confluence by a reduction and a relation represented by a dashed arrow and a dashed line, respectively:*

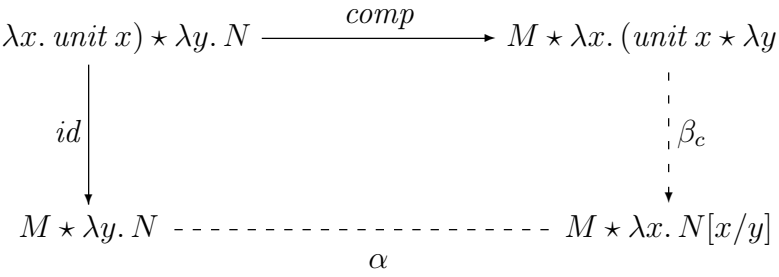


where $x \notin FV(N)$, which is the side condition to rule comp; therefore the two terms in the lower line of the diagram are syntactically identical.

Example 5.2. *In this example we see how outer reduction by comp, overlapping with outer id, can be recovered by an inner reduction by id:*



Example 5.3. *Here the outer reduction by comp overlaps with an inner reduction by id. This is recovered by means of an inner reduction by β_c:*



where $x \notin FV(N)$ as observed in Example 5.1, and therefore $\lambda x.\, N[x/y]$ is the renaming by x of the bound variable y in $\lambda y.\, N$: then the dashed line represents α-congruence.

After having inspected the above examples, one might by tempted to conclude that the reduction in Definition 4.1 enjoys the diamond property, namely it is confluent within (at most) two single steps, one per side (for the diamond property see (12) below: we say here 'at most' because $\longrightarrow$ is not reflexive). Unfortunately, this is not the case because of rule β_c, that can multiplicate redexes in the reduced term exactly as the β-rule in ordinary λ-calculus. Even worse, rule *comp* generates critical pairs with all other rules and with itself, preventing the simple extension of confluence proofs for β-reduction to succeed.

Following a strategy used in case of call-by-need calculi with the let-construct (see e.g. [11, 10]), we split the proof in three steps, proving confluence of $\beta_c \cup id$ and *comp* separately, and then combining these results by means of the commutativity of these relations.

In the first step we adapt the parallel reduction method, originally due to Tait and Martin Löf, and further developed by Takahashi [23]. See e.g. the book [24] ch. 10. Let's define the following relation $\multimap$:

Definition 5.4. *The relation $\multimap \subseteq Term \times Term$ is inductively defined by:*

i) $x \multimap x$

ii) $M \multimap N \Rightarrow \lambda x.M \multimap \lambda x.N$

iii) $V \multimap V' \Rightarrow unit\, V \multimap unit\, V'$

iv) $M \multimap M'$ *and* $V \multimap V' \Rightarrow M \star V \multimap M' \star V'$

v) $M \multimap M'$ *and* $V \multimap V' \Rightarrow unit\, V \star \lambda x.M \multimap M'[V'/x]$

vi) $M \multimap M' \Rightarrow M \star \lambda x.unit\, x \multimap M'$

By (i) - (iv) above, relation $\multimap$ is reflexive and coincides with its compatible closure. Also $\longrightarrow_{\beta_c,id} \subseteq \multimap$; intentionally, this is not the case w.r.t. the whole $\longrightarrow$.

Lemma 5.5. *For $M, M' \in Com$ and $V, V' \in Val$ and every variable x, if $M \multimap M'$ and $V \multimap V'$, then $M[V/x] \multimap M'[V'/x]$.*

Proof. By an easy induction on the definition of $M \multimap M'$ and $V \multimap V'$. □

Now, by means of Lemma 5.5 one easily proves that $\multimap \subseteq \xrightarrow{*}_{\beta_c,id}$.

The next step in the proof is to show that the relation $\multimap$ satisfies the *triangle property TP*:

$$\forall P\, \exists P^*\, \forall Q.\ P \multimap Q \ \Rightarrow\ Q \multimap P^* \tag{11}$$

where $P, P^*, Q \in \mathit{Term}$. TP implies the *diamond property DP*, which for $\multimap$ is:

$$\forall P, Q, R.\ P \multimap Q \ \&\ P \multimap R \ \Rightarrow\ \exists P'.\ Q \multimap P' \ \&\ R \multimap P' \tag{12}$$

In fact, if TP holds then we can take $P' \equiv P^*$ in DP, since the latter only depends on P. We then define P^* in terms of P as follows:

i) $x^* \equiv x$

ii) $(\lambda x.M)^* \equiv \lambda x.M^*$

iii) $(\mathit{unit}\, V)^* \equiv \mathit{unit}\, V^*$

iv) $(\mathit{unit}\, V \star \lambda x.M)^* \equiv M^*[V^*/x]$

v) $(M \star \lambda x.\mathit{unit}\, x)^* \equiv M^*$, if $M \not\equiv \mathit{unit}\, V$ for $V \in \mathit{Val}$

vi) $(M \star V)^* \equiv M^* \star V^*$, $M \not\equiv \mathit{unit}\, W$ for $W \in \mathit{Val}$ and $V \not\equiv \lambda x.\mathit{unit}\, x$

Lemma 5.6. *For all $P, Q \in \mathit{Term}$, if $P \multimap Q$ then $Q \multimap P^*$, namely $\multimap$ satisfies TP.*

Proof. By induction on $P \multimap Q$. The base case $x \multimap x$ follows by $x^* \equiv x$. All remaining cases follow by the induction hypotheses; in particular if $P \equiv \mathit{unit}\, V \star \lambda x.M \multimap M'[V'/x] \equiv Q$ because $M \multimap M'$ and $V \multimap V'$, then by induction $M' \multimap M^*$ and $V' \multimap V^*$, so that $M'[V'/x] \multimap M^*[V^*/x] \equiv P^*$ by Lem. 5.5. □

According to [21], Def. 3.1.11, a notion of reduction R is said to be *confluent* or *Church-Rosser*, shortly *CR*, if $\xrightarrow{*}_R$ satisfies *DP*; more explicitly for all $M, N, L \in \mathit{Com}$:

$$M \xrightarrow{*}_R N \ \&\ M \xrightarrow{*}_R L \Rightarrow \exists M' \in \mathit{Com}.\, N \xrightarrow{*}_R M' \ \&\ L \xrightarrow{*}_R M'$$

Corollary 5.7. *The notion of reduction $\beta_c \cup id$ is CR.*

Proof. As observed above $\longrightarrow_{\beta_c,id} \subseteq \multimap$, hence $M \xrightarrow{*}_{\beta_c,id} N$ implies $M \multimap^+ N$, where $\multimap^+$ is the transitive closure of $\multimap$, and similarly $M \multimap^+ L$. By Lemma 5.6 $\multimap$ satisfies *TP*, hence it satisfies *DP*. By an easy argument (see e.g. [21] Lemma 3.2.2) we conclude that $N \multimap^+ M'$ and $L \multimap^+ M'$ for some M', from which the thesis follows by the fact that $\multimap \subseteq \xrightarrow{*}_{\beta_c,id}$. □

To prove confluence of *comp* (more properly of its contextual closure) we use Newman Lemma (see [21], Prop. 3.1.24). A notion of reduction R is *weakly Church-Rosser*, shortly *WCR*, if for all $M, N, L \in Com$:

$$M \longrightarrow_R N \;\&\; M \longrightarrow_R L \Rightarrow \exists M' \in Com.\, N \xrightarrow{*}_R M' \;\&\; L \xrightarrow{*}_R M'$$

Lemma 5.8. *The notion of reduction comp is WCR.*

Proof. It suffices to show the thesis for the critical pair $M_1 \longrightarrow_{comp} M_2$ and $M_1 \longrightarrow_{comp} M_3$ where:

$$\begin{array}{lcl} M_1 & \equiv & ((L \star \lambda x.M) \star \lambda y, N) \star \lambda z.P \\ M_2 & \equiv & (L \star \lambda x.M) \star \lambda y.(N \star \lambda z.P) \\ M_3 & \equiv & (L \star \lambda x.(M \star \lambda y.N)) \star \lambda z.P \end{array}$$

Then in one step we have:

$$M_2 \longrightarrow_{comp} L \star \lambda x.(M \star \lambda y.(N \star \lambda z.P)) \equiv M_4$$

but

$$M_3 \longrightarrow_{comp} L \star \lambda x.((M \star \lambda y.N) \star \lambda z.P) \longrightarrow_{comp} M_4$$

where the two reduction steps are necessary. □

Recall that a notion of reduction R is *strongly normalizing*, shortly *SN*, if there exists no infinite reduction $M \longrightarrow_R M_1 \longrightarrow_R M_2 \longrightarrow_R \cdots$ out of any $M \in Com$.

Lemma 5.9. *The notion of reduction comp is SN.*

Proof. Given $M \in Com$ let's denote by the same M the expression obtained by marking differently all occurrences of $\star$ in M, say $\star_1, \ldots, \star_n$. We say that $\star_i$ is to the left to $\star_j$ in M if there exists a subterm $L \star_j V$ of M such that

$\star_i$ occurs in L. Finally, let's denote by $\sharp M$ the number of pairs $(\star_i, \star_j)$ such that $\star_i$ is to the left to $\star_j$ in M.

If a term includes an *comp*-redex $(L \star_i \lambda x.N) \star_j \lambda y.P$, which is contracted to $L \star_i \lambda x.(N \star_j \lambda y.P)$, then $\star_i$ is to the left to $\star_j$ in the redex, but not in the contractum. Also it is easily seen by induction on terms that, if $\star_i$ is not to the left to $\star_j$ in M and $M \longrightarrow_{comp} N$, the same holds in N.

It follows that, if $M \longrightarrow_{comp} N$ then $\sharp M > \sharp N$, hence *comp* is *SN*. □

Corollary 5.10. *The notion of reduction comp is CR.*

Proof. By Lem. 5.8, 5.9 and by Newman Lemma, stating that a notion of reduction which is *WCR* and *SN* is *CR*. □

Finally we show that $\longrightarrow_{\beta_c,id}$ and $\longrightarrow_{comp}$ commute. The following definitions are from [25], Def. 2.7.9. Two relations $\longrightarrow_1$ and $\longrightarrow_2$ over *Com* are said to *commute* if, for all M, N, L:

$$N \,{}_1\!\overset{*}{\longleftarrow} M \overset{*}{\longrightarrow}_2 L \Rightarrow \exists P \in Com.\ N \overset{*}{\longrightarrow}_2 P \,{}_1\!\overset{*}{\longleftarrow} L$$

Relations $\longrightarrow_1$ and $\longrightarrow_2$ *strongly commute* if, for all M, N, L:

$$N \,{}_1\!\longleftarrow M \longrightarrow_2 L \Rightarrow \exists P \in Com.\ N \overset{=}{\longrightarrow}_2 P \,{}_1\!\overset{*}{\longleftarrow} L$$

where $\overset{=}{\longrightarrow}_2$ is $\longrightarrow_2 \cup =$, namely at most one reduction step.

Lemma 5.11. *Reductions* $\longrightarrow_{\beta_c,id}$ *and* $\longrightarrow_{comp}$ *commute.*

Proof. By Lemma 2.7.11 in [25], two strongly commuting relations commute, and commutativity is clearly symmetric; hence it suffices to show that

$$N \,{}_{\beta_c,id}\!\longleftarrow M \longrightarrow_{comp} L \Rightarrow \exists P \in Com.\ N \overset{=}{\longrightarrow}_{comp} P \,{}_{\beta_c,id}\!\overset{*}{\longleftarrow} L.$$

We can limit the cases to the critical pairs, that are exactly those in examples 5.1, 5.2 and 5.3, which commute. □

Theorem 5.12 (Confluence)**.** *The notion of reduction* $\lambda\boldsymbol{C} = \beta_c \cup id \cup comp$ *is CR.*

Proof. By the commutative union lemma (see [25], Lem. 2.7.10 and [21], Prop. 3.3.5, where it is called Hindley-Rosen Lemma), if $\longrightarrow_{\beta_c,id}$ and $\longrightarrow_{comp}$ and are both *CR* (Cor. 5.7 and 5.10), and commute (Lem. 5.11), then $\longrightarrow_{\lambda\mathbf{C}} = \longrightarrow_{\beta_c,id} \cup \longrightarrow_{comp}$ is *CR*. □

Consequence of Theorem 5.12 is the unicity of the normal form of a term, if any. Also, by construction and Proposition 3.4, convertible terms, related by the reflexive, symmetric and transitive closure $=_{\lambda\mathbf{C}}$ of $\longrightarrow_{\lambda\mathbf{C}}$, namely its *convertibility relation*, are equated in any model of λ_c^u.

6. Intersection types for λ_c^u

Intersection types are an extension of Curry's simple type assignment system to untyped λ-terms, obtained by adding new types $\sigma \wedge \sigma'$ to be assigned to terms that have both type σ and σ'. Intersection types assignment systems form a whole family in the literature; of special interest to us is the system in [26], usually called BCD: see [13] part III. In BCD is introduced a notion of subtyping together with a universal type ω, that can be assigned to any term, leading to a notion called below *type theory*.

Definition 6.1 (Intersection types and Type theories)**.** *A* language of intersection types $\mathcal{T}$ *is a set of expressions* $\sigma, \sigma', \ldots$ *including a constant* ω *and closed under the intersection type constructor:* $\sigma \wedge \sigma'$*.*

An intersection type theory *(shortly a* type theory*) is a pair* $Th = (\mathcal{T}, \leq)$ *where* $\mathcal{T}$ *is a language of intersection types and* $\leq$ *a pre-order over* $\mathcal{T}$ *such that* ω *is the top,* $\wedge$ *is idempotent and commutative, and*

$$\sigma \wedge \sigma' \leq \sigma, \qquad \frac{\sigma \leq \sigma' \quad \sigma \leq \sigma''}{\sigma \leq \sigma' \wedge \sigma''}$$

A type theory is a presentation of a meet-semilattice with ω as top element; in particular $\wedge$ turns out to be monotonic. Different type theories give rise to different structures and hence to different type systems. We adapt BCD type theory to the case of λ_c^u, where two sorts of types correspond to the two sorts of terms.

Definition 6.2 (Intersection types for values and computations)**.** *Let TypeVar be a countable set of* type variables*, ranged over by* α*:*

$$\begin{array}{rrcll} \mathit{ValType}: & \delta & ::= & \alpha \mid \delta \to \tau \mid \delta \wedge \delta \mid \omega_{\mathsf{V}} & \text{(value types)} \\ \mathit{ComType}: & \tau & ::= & T\delta \mid \tau \wedge \tau \mid \omega_{\mathsf{C}} & \text{(computation types)} \end{array}$$

Intersection types from Def. 6.2 are better understood as predicates of values and computations, respectively, or as refinement types of the two types of λ_c^u, that is, using the notation in [27], $\delta \sqsubset D = D \to TD$ in case of values, and $\tau \sqsubset TD$ in case of computations.

In the definition of language *ValType* and consequently *ComType* the set of *TypeVar* (also called *atoms*) is left unspecified and it is a parameter.

Definition 6.3 (Type theories Th_{V} and Th_{C})**.** *The intersection type theories* $Th_{\mathsf{V}} = (ValType, \leq_{\mathsf{V}})$ *and* $Th_{\mathsf{C}} = (ComType, \leq_{\mathsf{C}})$ *are the least type theories such that:*

$$\delta \leq_{\mathsf{V}} \omega_{\mathsf{V}} \qquad \omega_{\mathsf{V}} \leq_{\mathsf{V}} \omega_{\mathsf{V}} \to \omega_{\mathsf{C}}$$

$$(\delta \to \tau) \wedge (\delta \to \tau') \leq_{\mathsf{V}} \delta \to (\tau \wedge \tau') \qquad \frac{\delta' \leq_{\mathsf{V}} \delta \quad \tau \leq_{\mathsf{C}} \tau'}{\delta \to \tau \leq_{\mathsf{V}} \delta' \to \tau'}$$

$$\tau \leq_{\mathsf{C}} \omega_{\mathsf{C}} \qquad T\delta \wedge T\delta' \leq_{\mathsf{C}} T(\delta \wedge \delta') \qquad \frac{\delta \leq_{\mathsf{V}} \delta'}{T\delta \leq_{\mathsf{C}} T\delta'}$$

Remark 6.4. *Writing* $=_{\mathsf{V}}$ *for* $\leq_{\mathsf{V}} \cap \leq_{\mathsf{V}}^{-1}$ *and similarly* $=_{\mathsf{C}}$*, we see that all the axioms but* $\delta \leq_{\mathsf{V}} \omega_{\mathsf{V}}$ *and* $\tau \leq_{\mathsf{C}} \omega_{\mathsf{C}}$ *are actually equalities. In particular, if* $\tau \neq_{\mathsf{C}} \omega_{\mathsf{C}}$ *then for a finite set of* δ_i *we have* $\tau =_{\mathsf{C}} \bigwedge_i T\delta_i =_{\mathsf{C}} T(\bigwedge_i \delta_i)$*.*

Type theories Th_{V} and Th_{C} depend on each other. Except for the distinction among value and computation types, theory Th_{V} is exactly the same as the type theory of BCD. Theory Th_{C} treats T as a type modality and as a morphism of meet-semilattices: hence it is monotonic and preserves meets. An important feature is that $T\omega_{\mathsf{V}}$ is strictly less than ω_{C} in general; this is consistent with the interpretation of $T\omega_{\mathsf{V}}$ as the largest type of convergent terms. Indeed in Corollary 8.8 we shall prove that $\omega_C \leq_{\mathsf{C}} T\omega_{\mathsf{V}}$ is not derivable from the system in Definition 6.3 by exhibiting a type interpretation such that this inequation doesn't hold.

Lemma 6.5. *If* $\tau \in ComType$ *is such that* $\tau \neq_{\mathsf{C}} \omega_{\mathsf{C}}$ *then for some* $\delta \in ValType$ *we have* $\tau =_{\mathsf{C}} T\delta$*; hence* $\tau \leq_{\mathsf{C}} T\omega_{\mathsf{V}}$*.*

Proof. By induction over τ. The only non trivial case is when $\tau \equiv \tau_1 \wedge \tau_2$. From $\tau_1 \wedge \tau_2 \neq_{\mathsf{C}} \omega_{\mathsf{C}}$ it follows that at least one of them is different than ω_{C}: if say $\tau_1 =_{\mathsf{C}} \omega_{\mathsf{C}}$ then $\tau_1 \wedge \tau_2 =_{\mathsf{C}} \tau_2 \neq_{\mathsf{C}} \omega_{\mathsf{C}}$ so that $\tau_2 =_{\mathsf{C}} T\delta_2$ by induction. Finally, if both τ_1 and τ_2 are not equated to ω_{C} then by induction $\tau_1 \wedge \tau_2 =_{\mathsf{C}} T\delta_1 \wedge T\delta_2 =_{\mathsf{C}} T(\delta_1 \wedge \delta_2) \leq_{\mathsf{C}} T\omega_{\mathsf{V}}$, for some $\delta_1, \delta_2 \in ValType$. □

We are now in place to introduce the type assignment system for λ_c^u.

Definition 6.6 (Type assignment)**.** *A* basis *is a finite set of typings* $\Gamma = \{x_1 : \delta_1, \ldots x_n : \delta_n\}$ *with pairwise distinct variables* x_i*, whose* domain *is the set* $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$*. A basis determines a function from variables to types such that* $\Gamma(x) = \delta$ *if* $x : \delta \in \Gamma$*,* $\Gamma(x) = \omega_{\mathsf{V}}$ *otherwise.*

A judgment *is an expression of either shapes:* $\Gamma \vdash V : \delta$ *or* $\Gamma \vdash M : \tau$*. It is* derivable *if it is the conclusion of a derivation according to the rules:*

$$\frac{x : \delta \in \Gamma}{\Gamma \vdash x : \delta}\,(Ax) \qquad \frac{\Gamma, x : \delta \vdash M : \tau}{\Gamma \vdash \lambda x.M : \delta \to \tau}\,(\to I)$$

$$\frac{\Gamma \vdash V : \delta}{\Gamma \vdash \mathit{unit}\, V : T\delta}\,(\mathit{unit}\ I) \qquad \frac{\Gamma \vdash M : T\delta \quad \Gamma \vdash V : \delta \to \tau}{\Gamma \vdash M \star V : \tau}\,(\to E)$$

where $\Gamma, x : \delta = \Gamma \cup \{x : \delta\}$ *with* $x : \delta \notin \Gamma$*, and the rules:*

$$\frac{}{\Gamma \vdash P : \omega}\ (\omega) \quad \frac{\Gamma \vdash P : \sigma \quad \Gamma \vdash P : \sigma'}{\Gamma \vdash P : \sigma \wedge \sigma'}\ (\wedge I) \quad \frac{\Gamma \vdash P : \sigma \quad \sigma \leq \sigma'}{\Gamma \vdash P : \sigma'}\ (\leq)$$

where either $P \in \mathit{Val}$*,* $\omega \equiv \omega_{\mathsf{V}}$*,* $\sigma, \sigma' \in \mathit{ValType}$ *and* $\leq = \leq_{\mathsf{V}}$ *or* $P \in \mathit{Com}$*,* $\omega \equiv \omega_{\mathsf{C}}$*,* $\sigma, \sigma' \in \mathit{ComType}$ *and* $\leq = \leq_{\mathsf{C}}$*.*

In the proof texts we write $\Gamma \vdash V : \delta$ and $\Gamma \vdash M : \tau$ to mean that these judgments are derivable. Because of rule (ω), it is not true in general that if $\Gamma \vdash P : \sigma$ then $FV(P) \subseteq \mathrm{dom}\,(\Gamma)$; however under the same hypothesis we have that $\Gamma \restriction FV(P) \vdash P : \sigma$, where, for $X \subseteq \mathit{Var}$, $\Gamma \restriction X = \{x : \delta \mid x : \delta \in \Gamma \ \&\ x \in X\}$ (the restriction of Γ to X).

Among the elementary properties of the system, we state the admissibility of the following rules.

Lemma 6.7 (Weakening and Strengthening)**.** *The following rules are admissible:*

$$\frac{\Gamma \vdash P : \sigma \quad \Gamma \subseteq \Gamma'}{\Gamma' \vdash P : \sigma}\,(W) \qquad \frac{\Gamma, x : \delta \vdash P : \sigma \quad \delta' \leq_{\mathsf{V}} \delta}{\Gamma, x : \delta' \vdash P : \sigma}\,(S)$$

where $P \in \mathit{Term}$ *and* σ *is either in ValType or in ComType according to the sort of* P*.*

Proof. Admissibility of rule (W) is proved by a straightforward induction over the derivation of $\Gamma \vdash P : \sigma$. Concerning rule (S) we also reason by induction over the derivation of $\Gamma, x : \delta \vdash P : \sigma$, where x is not bound in P as it is element of $\mathrm{dom}\,(\Gamma, x : \delta)$. It follows that in the derivation the only judgements in which x is the subject of the right hand side typing are either

instances of rule $(\leq)$ or instances of (Ax), that is of the shape $\Gamma', x : \delta \vdash x : \delta$ for some $\Gamma' \supseteq \Gamma$. In the first case the thesis follows by induction. In the second case we obtain a new reduction with the same conclusion by replacing (Ax) by the inference

$$\dfrac{\dfrac{}{\Gamma', x : \delta' \vdash x : \delta'}\,(Ax) \qquad \delta' \leq_{\mathsf{V}} \delta}{\Gamma', x : \delta' \vdash x : \delta}\,(\leq)$$

□

7. Subject reduction and expansion

In this section we establish the minimal requirement for a sound type system, namely that types are preserved by reductions. Moreover, we prove that types are preserved by subject expansion, which is a characteristic property of intersection type systems with universal type ω.

The next lemma is an extension of the analogous property of BCD type system, also called Inversion Lemma in [13] 14A.1.

Lemma 7.1 (Generation lemma)**.** *Assume that $\delta \neq \omega_{\mathsf{V}}$ and $\tau \neq \omega_{\mathsf{C}}$, then:*

i) $\Gamma \vdash x : \delta \Leftrightarrow \Gamma(x) \leq_{\mathsf{V}} \delta$

ii) $\Gamma \vdash \lambda x.M : \delta \Leftrightarrow$
$\exists I, \delta_i, \tau_i.\ \forall i \in I.\ \Gamma, x : \delta_i \vdash M : \tau_i \ \ \&\ \ \bigwedge_{i\in I} \delta_i \to \tau_i \leq_{\mathsf{V}} \delta$

iii) $\Gamma \vdash \mathit{unit}\, V : \tau \Leftrightarrow \exists I, \delta_i\ \forall i \in I.\ \Gamma \vdash V : \delta_i \ \ \&\ \ \bigwedge_{i\in I} T\delta_i \leq_{\mathsf{C}} \tau$

iv) $\Gamma \vdash M \star V : \tau \Leftrightarrow$
$\exists I, \delta_i, \tau_i.\ \forall i \in I.\ \Gamma \vdash M : T\delta_i \ \ \&\ \ \Gamma \vdash V : \delta_i \to \tau_i \ \ \&\ \ \bigwedge_{i\in I} \tau_i \leq_{\mathsf{C}} \tau$

Proof. The implications $\Leftarrow$ are immediate. To see the implications $\Rightarrow$ we reason by induction over the derivations, by distinguishing the cases of the last rule. Parts i) and ii) are the same as for ordinary intersection types and λ-calculus; part iii) is immediate by the induction hypothesis, hence we treat part iv) only.

If the last rule in the derivation of $\Gamma \vdash M \star V : \tau$ is $(\to \mathrm{E})$ just take I as a singleton set. If it is $(\leq)$ then the thesis follows immediately by induction and the transitivity of $\leq_{\mathsf{C}}$. Finally, suppose that the derivation ends by

$$\dfrac{\Gamma \vdash M \star V : \tau' \quad \Gamma \vdash M \star V : \tau''}{\Gamma \vdash M \star V : \tau' \wedge \tau''}\,(\wedge I)$$

and $\tau \equiv \tau' \wedge \tau''$. Then by induction we have

$$\exists I, \delta'_i, \tau'_i.\ \forall i \in I.\ \Gamma \vdash M : T\delta'_i \ \ \&\ \ \Gamma \vdash V : \delta'_i \to \tau'_i \ \ \&\ \ \bigwedge_{i\in I} \tau'_i \leq_{\mathsf{C}} \tau'$$

and

$$\exists J, \delta''_j, \tau''_j.\ \forall j \in J.\ \Gamma \vdash M : T\delta''_j \ \ \&\ \ \Gamma \vdash V : \delta''_j \to \tau''_j \ \ \&\ \ \bigwedge_{j\in J} \tau''_j \leq_{\mathsf{C}} \tau''$$

From this the thesis immediately follows by observing that

$$\bigwedge_{i\in I} \tau'_i \leq_{\mathsf{C}} \tau' \ \ \&\ \ \bigwedge_{j\in J} \tau''_j \leq_{\mathsf{C}} \tau'' \Rightarrow \bigwedge_{i\in I} \tau'_i \wedge \bigwedge_{j\in J} \tau''_j \leq_{\mathsf{C}} \tau' \wedge \tau''.$$

□

We observe that the statements of Lem. 7.1 could be stronger, since whenever we say that if $\Gamma \vdash P : \sigma$ then $\Gamma' \vdash Q : \sigma'$ it is always the case that the derivation of the latter judgment is a subderivation of the former. Furthermore the inverse of all implications hold.

The following lemma, necessary to the subsequent proofs, establishes a fundamental property of intersection type theories including arrow types, known as *extended applicative type structures*, or EATS [28]. It has been stated the first time in [26], 2.4 (ii), and it has been widely covered for intersection type theories in [13] with the name β-soundness (Definition 14A.4).

Lemma 7.2. *Let $\tau \neq_{\mathsf{C}} \omega_{\mathsf{C}}$, then:*

$$\bigwedge_{i\in I}(\delta_i \to \tau_i) \leq_{\mathsf{V}} \delta \to \tau \Leftrightarrow \exists J \subseteq I.\ J \neq \emptyset \ \ \&\ \ \delta \leq_{\mathsf{V}} \bigwedge_{j\in J} \delta_j \ \ \&\ \ \bigwedge_{j\in J} \tau_j \leq_{\mathsf{C}} \tau$$

Proof. By induction over the definition of $\leq_{\mathsf{V}}$ and $\leq_{\mathsf{C}}$. □

Lemma 7.3 (Substitution lemma)**.** *If $\Gamma, x : \delta \vdash M : \tau$ and $\Gamma \vdash V : \delta$ then $\Gamma \vdash M[V/x] : \tau$.*

Proof. By induction over the derivation of $\Gamma, x : \delta \vdash M : \tau$. For the induction to go through, one has to show the auxiliary statement that if $\Gamma, x : \delta \vdash W : \delta'$ for some $W \in \mathit{Val}$ then $\Gamma \vdash W[V/x] : \delta'$. Details are routine. □

Theorem 7.4 (Subject reduction)**.** *If $\Gamma \vdash M : \tau$ and $M \longrightarrow N$ then $\Gamma \vdash N : \tau$.*

Proof. Let us assume that $\tau \neq \omega_{\mathsf{C}}$ since the thesis is trivial otherwise. The proof is by induction over the definition of $M \longrightarrow N$, using Lem. 7.1. We treat just the interesting cases.

Case (β_c): then $M \equiv \mathit{unit}\ V \star (\lambda x.M')$ and $N \equiv M'[V/x]$. From the hypothesis $\Gamma \vdash M : \tau$, by iii) and iv) of Lem. 7.1 we have that there exist a finite set I and types δ_i, δ'_i and τ_i for all $i \in I$ such that:

1. $\Gamma \vdash V : \delta'_i$ with $\delta'_i \leq \delta_i$;
2. $\Gamma \vdash \lambda x.M' : \delta_i \to \tau_i$ with $\bigwedge_{i\in I} \tau_i \leq_{\mathsf{C}} \tau$

By ii) of the same lemma for all $i \in I$ there is J_i such that for all $j \in J_i$:

3. $\Gamma, x : \delta_{ij} \vdash M : \tau_{ij}$ with $\bigwedge_{j\in J_i} \delta_{ij} \to \tau_{ij} \leq_{\mathsf{V}} \delta_i \to \tau_i$

In virtue of Lem. 7.2, we may assume w.l.o.g that the non empty J_i have been chosen so that $\delta_i \leq_{\mathsf{V}} \bigwedge_{j\in J_i} \delta_{ij}$ and $\bigwedge_{j\in J_i} \tau_{ij} \leq_{\mathsf{C}} \tau_i$ for all $i \in I$.

It follows that $\delta'_i \leq_{\mathsf{V}} \delta_i \leq_{\mathsf{V}} \delta_{ij}$ for all i and j so that by (1) we have $\Gamma \vdash V : \delta_{ij}$ by rule $(\leq)$. It follows by Lem. 7.3 and (1) that $\Gamma \vdash M'[V/x] : \tau_{ij}$; now $\bigwedge_{i\in I} \bigwedge_{j\in J_i} \tau_{ij} \leq_{\mathsf{C}} \tau$, so that $\Gamma \vdash M'[V/x] : \tau$ by repeated applications of rule $(\wedge I)$ and $(\leq)$.

Case (comp): then $M \equiv (L \star \lambda x.M') \star \lambda y.N'$ and $N \equiv L \star \lambda x.(M' \star \lambda y.N')$ where $x \notin FV(N')$. As before from $\Gamma \vdash M : \tau$ and Lem. 7.1 we know that there exist I, δ_i, τ_i such that for all $i \in I$:

4. $\Gamma \vdash L \star \lambda x.M' : T\delta_i$
5. $\Gamma \vdash \lambda y.N' : \delta_i \to \tau_i$ with $\bigwedge_{i\in I} \tau_i \leq_{\mathsf{C}} \tau$

From (4) it follows that for all $i \in I$ there are $J_i, \delta_{ij}, \tau_{ij}$ such that for all $j \in J_i$:

6. $\Gamma \vdash L : T\delta_{ij}$
7. $\Gamma \vdash \lambda x.M' : \delta_{ij} \to \tau_{ij}$ with $\bigwedge_{j\in J_i} \tau_{ij} \leq_{\mathsf{C}} T\delta_i$

Reasoning as in case (β_c), we obtain from (7) that for all $j \in J_i$ there exist $K_j, \delta_{ijk}, \tau_{ijk}$ s.t.

8. $\Gamma, x : \delta_{ijk} \vdash M' : \tau_{ijk}$ with $\bigwedge_{k\in K_j} \delta_{ijk} \to \tau_{ijk} \leq_{\mathsf{V}} \delta_{ij} \to \tau_{ij}$

Assuming as before that the J_i and the K_j have been suitably chosen, by Lem. 7.2 we have that

9. $\delta_i \leq_{\mathsf{V}} \bigwedge_{j\in J_i} \delta_{ij}$ and $\bigwedge_{j\in J_i} \tau_{ij} \leq_{\mathsf{C}} T\delta_i$
10. $\delta_{ij} \leq_{\mathsf{V}} \bigwedge_{k\in K_j} \delta_{ijk}$ and $\bigwedge_{k\in K_j} \tau_{ijk} \leq_{\mathsf{C}} \tau_{ij}$

Therefore, for all i, j, k we have $\delta_{ij} \leq_{\mathsf{V}} \delta_{ijk}$ and $\tau_{ijk} \leq_{\mathsf{C}} T\delta_i$; hence from (8) by (S) and $(\leq)$ we have $\Gamma, x : \delta_{ij} \vdash M' : T\delta_i$. On the other hand by admissibility of rule (W), from (5) and the fact that $x \notin FV(N')$, we have that $\Gamma, x : \delta_{ij} \vdash \lambda y.N' : \delta_i \to \tau_i$. Hence for all i:

$$\dfrac{\Gamma \vdash L : T\delta_{ij} \qquad \dfrac{\dfrac{\Gamma, x : \delta_{ij} \vdash M' : T\delta_i \quad \Gamma, x : \delta_{ij} \vdash \lambda y.N' : \delta_i \to \tau_i}{\Gamma, x : \delta_{ij} \vdash M' \star \lambda y.N' : \tau_i}}{\Gamma \vdash \lambda x.(M' \star \lambda y.N') : \delta_{ij} \to \tau_i}}{\Gamma \vdash L \star \lambda x.(M' \star \lambda y.N') : \tau_i}$$

Now the thesis follows by rules $(\wedge I)$ and $(\leq)$, using $\bigwedge_{i\in I} \tau_i \leq_{\mathsf{C}} \tau$.

Case *id* is immediate from Lem. 7.1; all cases dealing with the compatible closure are easy consequences of the induction hypotheses. □

Toward the proof of subject expansion, we need a lemma that is the inverse of Lem. 7.3.

Lemma 7.5 (Expansion Lemma)**.** *If $\Gamma \vdash P[V/x] : \sigma$ with $V \in Val$, $P \in Term$ and either $\sigma \in ValType$ or $\sigma \in ComType$ according to the sort of P, then then there exists $\delta \in ValType$ such that:*

$$\Gamma \vdash V : \delta \quad \textit{and} \quad \Gamma, x : \delta \vdash P : \sigma$$

Proof. If σ is either ω_{V} or ω_{C} then the thesis is trivial. Otherwise, let us assume w.l.o.g. that $x \notin FV(P[V/x]) \cup \mathrm{dom}\,(\Gamma)$ and that V is free for x in P, that is $FV(V) \cap BV(P) = \emptyset$, which is a necessary condition for the substitution $P[V/x]$ to be capture avoiding. Then we proceed by induction over $P[V/x]$, and by cases of P.

Case $P \equiv x$: then σ is some $\delta \in ValType$; since clearly $x[V/x] \equiv V$ then $\Gamma \vdash V : \delta$ by the hypothesis, and $\Gamma, x : \delta \vdash x : \delta$ by (Ax), where $\Gamma, x : \delta$ is a basis since $x \notin \mathrm{dom}\,(\Gamma)$.

Case $P \equiv y \not\equiv x$: then we trivially have $y[V/x] \equiv y$, so that we obtain the thesis by taking $\delta \equiv \omega_{\mathsf{V}}$.

Case $P \equiv \lambda y.M$: then $P[V/x] \equiv \lambda y.(M[V/x])$; in particular since free variables in V cannot be caught in $P[V/x]$ by the binding λy, we freely assume that $y \notin FV(V)$. From the hypothesis $\Gamma \vdash \lambda y.(M[V/x]) : \sigma$, by ii) of Lem. 7.1, it follows that there exist I and δ_i, τ_i such that $\Gamma, y : \delta_i \vdash M[V/x] : \tau_i$ for all $i \in I$ and $\bigwedge_{i\in I} \delta_i \to \tau_i \leq_{\mathsf{V}} \sigma$.

By induction for all $i \in I$ there exist δ'_i such that $\Gamma, y : \delta_i \vdash V : \delta'_i$ and $\Gamma, y : \delta_i, x : \delta'_i \vdash M : \tau_i$. Since $y \notin FV(V)$, from $\Gamma, y : \delta_i \vdash V : \delta'_i$ we obtain $\Gamma \vdash V : \delta'_i$. Taking $\delta = \bigwedge_{i\in I} \delta'_i$ we obtain $\Gamma \vdash V : \delta$ by rule $(\wedge I)$ and $\Gamma, y : \delta_i, x : \delta \vdash M : \tau_i$ by (S). ; on the other hand we get $\Gamma, x : \delta \vdash \lambda y.M : \delta_i \to \tau_i$ for all $i \in I$ by rule $(\to \mathrm{I})$ and $(\wedge I)$. Hence we conclude by rule $(\leq)$.

Case $P \equiv \mathit{unit}\, W$: then $P[V/x] \equiv \mathit{unit}\, (W[V/x])$ and the thesis follows by iii) of Lem. 7.1 and the induction hypothesis.

Case $P \equiv M \star W$: then $P[V/x] \equiv (M[V/x]) \star (W[V/x])$. By iv) of Lem. 7.1 and induction, there exist I and δ_i, τ_i and δ'_1, δ'_2 such that $\Gamma \vdash V : \delta'_j$, $\Gamma, x : \delta'_j \vdash M : T\delta_i$ and $\Gamma, x : \delta'_j \vdash W : \delta_i \to \tau_i$ for all $i \in I$ and $j = 1, 2$, such that $\bigwedge_{i\in I} \tau_i \leq_{\mathsf{C}} \sigma$. Take $\delta = \delta'_1 \wedge \delta'_2$: then $\Gamma \vdash V : \delta$ by $(\wedge I)$ and $\Gamma, x : \delta \vdash M : T\delta_i$ and $\Gamma, x : \delta \vdash W : \delta_i \to \tau_i$ for all $i \in I$ by (S). Now $\Gamma, x : \delta \vdash M \star W : \sigma$ follows by $(\to \mathrm{E})$, $(\wedge I)$ and $(\leq)$.

□

Theorem 7.6 (Subject expansion)**.** *If* $\Gamma \vdash N : \tau$ *and* $M \longrightarrow N$ *then* $\Gamma \vdash M : \tau$.

Proof. The proof is by induction over $M \longrightarrow N$, assuming that $\tau \neq_{\mathsf{C}} \omega_{\mathsf{C}}$. The only interesting cases are the following.

Case (β_c): then $M \equiv \mathit{unit}\, V \star (\lambda x.M')$ and $N \equiv M'[V/x]$. By Lem. 7.5 there exists δ such that $\Gamma \vdash V : \delta$ and $\Gamma, x : \delta \vdash M' : \tau$. Then $\Gamma \vdash \mathit{unit}\, V : T\delta$ by rule (unit I) and $\Gamma \vdash \lambda x.M' : \delta \to \tau$ by rule $(\to \mathrm{I})$. We conclude that $\Gamma \vdash \mathit{unit}\, V \star (\lambda x.M') : \tau$ by rule $(\to \mathrm{E})$.

Case (comp): then $M \equiv (L \star \lambda x.M') \star \lambda y.N'$ and $N \equiv L \star \lambda x.(M' \star \lambda y.N')$.

By iv) of Lem. 7.1, from $\Gamma \vdash N : \tau$ there exist I, δ_i, τ_i such that for all $i \in I$ we have $\Gamma \vdash L : T\delta_i$, $\Gamma \vdash \lambda x.(M' \star \lambda y.N') : \delta_i \to \tau_i$ and $\bigwedge_{i\in I} \tau_i \leq_{\mathsf{C}} \tau$. By ii) of Lem. 7.1 for all $i \in I$ there exist $J_i, \delta_{ij}, \tau_{ij}$ such that $\Gamma, x : \delta_{ij} \vdash M' \star \lambda y.N' : \tau_{ij}$ and $\bigwedge_{j\in J_i} \delta_{ij} \to \tau_{ij} \leq_{\mathsf{V}} \delta_i \to \tau_i$. From this and by iv) of Lem. 7.1, for all $j \in J_i$ there are $K_j, \delta_{ijk}, \tau_{ijk}$ such that for all $k \in K_j$ we have $\Gamma, x : \delta_{ij} \vdash M' : T\delta_{ijk}$ and $\Gamma, x : \delta_{ij} \vdash \lambda y.N' : \delta_{ijk} \to \tau_{ijk}$ and $\bigwedge_{k\in K_j} \tau_{ijk} \leq_{\mathsf{C}} \tau_{ij}$.

Now, by Lem. 7.2, from $\bigwedge_{j\in J_i} \delta_{ij} \to \tau_{ij} \leq_{\mathsf{V}} \delta_i \to \tau_i$ we have that there exists $\emptyset \neq J'_i \subseteq J_i$ such that for all $j \in J'_i$, $\delta_i \leq_{\mathsf{V}} \delta_{ij}$ and $\tau_{ij} \leq_{\mathsf{C}} \tau_i$. Hence we obtain that $T\delta_i \leq_{\mathsf{C}} T\delta_{ij}$ so that $\Gamma \vdash L : T\delta_{ij}$, for all i and the appropriate j. On the other hand by rule $(\to \mathrm{I})$ we know that $\Gamma \vdash \lambda x.M' : \delta_{ij} \to T\delta_{ijk}$, so that by rule $(\to \mathrm{E})$ we deduce $\Gamma \vdash L \star \lambda x.M' : T\delta_{ijk}$, for all i and the appropriate j, k.

From $\Gamma, x : \delta_{ij} \vdash \lambda y.N' : \delta_{ijk} \to \tau_{ijk}$ and the fact that $x \notin FV(\lambda y.N')$ we have that $\Gamma \vdash \lambda y.N' : \delta_{ijk} \to \tau_{ijk}$, that combined with the above, yields $\Gamma \vdash (L \star \lambda x.M') \star \lambda y.N' : \tau_{ijk}$ by rule $(\to \mathrm{E})$. But we know that $\bigwedge_{j\in J'_i, k\in K_j} \tau_{ijk} \leq_{\mathsf{C}} \bigwedge_{j\in J'_i} \tau_{ij} \leq_{\mathsf{C}} \tau_i$, for all $i \in I$: thus the thesis follows by $(\wedge I)$ and $(\leq)$ since $\bigwedge_{i\in I} \tau_i \leq_{\mathsf{C}} \tau$.

□

8. Convergence

Operational semantics of λ-calculi, as well as of programming languages, consists of giving meaning to terms via a definition of their execution. This can be done either by a small-step reduction relation (more precisely by considering some reduction strategy), or via an evaluation relation of terms to their values, often called *convergence* predicate. Both are examples of structural operational semantics in Plotkin's sense [29], but serve different purposes. Instead of describing the evaluation process in detail, which is also defined on open (sub)-terms, convergence is a relation among "programs", that are closed terms, and their equally closed values. Having treated reduction for λ^u_c, we now introduce a convergence predicate, whose definition is inspired by the analogous relation for the call-by-value λ-calculus.

Henceforth in this section, terms are closed if not otherwise stated. Let Com^0 and Val^0 be the set of closed computations and values, respectively.

Definition 8.1 (Convergence)**.** *The* convergence relation $\Downarrow \subseteq Com^0 \times \mathbb{N} \times Val^0$*, is defined as follows:*

$$unit\, V \Downarrow_0 V \qquad \frac{M \Downarrow_m V' \qquad N[V'/x] \Downarrow_n V}{M \star (\lambda x.N) \Downarrow_{m+n+1} V}$$

Notation:

$$\begin{array}{lcl} M \Downarrow V & \Leftrightarrow & \exists n.\, M \Downarrow_n V \\ M \Downarrow & \Leftrightarrow & \exists V.\, M \Downarrow V \end{array}$$

The notation $M \Downarrow_n V$ could have been written as $M \Downarrow_n unit\, V$, in the sense that a term of the shape $unit\, V$ is a sort of weak normal form of M w.r.t. the reduction $\longrightarrow$ which is defined among computations, not among computation and values. By choosing the definition above we intend to emphasize that $unit\, V$ is the "trivial" computation of V, that is terminated; in the context of λ_c this is not the same as V, and we want to save the idea that convergence relates programs, namely computations, to their results, that are values. The two concepts are related as stated in the lemma:

Lemma 8.2. $M \Downarrow V \Rightarrow M \xrightarrow{*} unit\, V$

Proof. By hypotheses $M \Downarrow V$, that is $M \Downarrow_n V$ for some $n \in \mathbb{N}$. Then we reason by induction over n. If $n = 0$ then $M \equiv unit\ V$ and trivially $unit\, V \xrightarrow{*} unit\, V$.

Otherwise $M \equiv M' \star \lambda y.N \Downarrow_n V$ and for some $W \in Val^0$:

$$\frac{M' \Downarrow_p W \qquad N[W/y] \Downarrow_q V}{M' \star \lambda y.\, N \Downarrow_n V} \quad \text{where } n = p + q + 1$$

By induction $M' \xrightarrow{*} unit\, W$ and $N[W/y] \xrightarrow{*} unit\, V$ so that

$$M' \star \lambda y.\, N \xrightarrow{*} unit\, W \star \lambda y.\, N \longrightarrow N[W/y] \xrightarrow{*} unit\, V.$$

□

The inverse implication does not hold. Indeed if $N \longrightarrow N'$ then $unit\ (\lambda x.\, N) \longrightarrow unit\ (\lambda x.\, N')$, that is $\lambda x.\, N'$ is a more refined value than $\lambda x.\, N$; however, $unit\ (\lambda x.\, N) \not\Downarrow \lambda x.\, N'$. We can only prove the weaker statement in the next lemma, which nonetheless suffices.

Lemma 8.3. *$M \stackrel{*}{\longrightarrow} N$ and $N \Downarrow V \;\Rightarrow\; \exists W.\ M \Downarrow W$ and $unit\ W \stackrel{*}{\longrightarrow} unit\ V$.*

Proof. By induction over the length of $M \stackrel{*}{\longrightarrow} N$. If it is 0 then $M \equiv N$ and the thesis is trivial. Suppose that $M \longrightarrow N' \stackrel{*}{\longrightarrow} N$ for some N'. Then by induction $N' \Downarrow W'$ for some W' s.t. $unit\ W' \stackrel{*}{\longrightarrow} unit\ V$. Although the lemma is about closed terms, when going through the cases we have to deal with open terms as well; therefore we strengthen the thesis as follows. Let $[\vec{U}/\vec{x}]$ be the simultaneous substitution of all variables $\vec{x}$ by the closed values $\vec{U}$. Then we show:

> if $M[\vec{U}/\vec{x}] \stackrel{*}{\longrightarrow} N[\vec{U}/\vec{x}]$ and $N[\vec{U}/\vec{x}] \Downarrow V$ for some substitution $[\vec{U}/\vec{x}]$ closing both M and N then $M[\vec{U}/\vec{x}] \Downarrow W$ and $unit\ W \stackrel{*}{\longrightarrow} unit\ V$ for some W.

To avoid using heavy notation, we shall keep implicit the reference to the substitution $[\vec{U}/\vec{x}]$ in all cases but in the last one, where it is needed.

In the following if $V \equiv \lambda x.P$ then we abbreviate $V[W] \equiv P[W/x]$, and similarly for other values that are not a variable. The proof proceeds by a secondary induction over the definition of $M \longrightarrow N'$.

$M \equiv unit\ W \star V' \longrightarrow V'[W] \equiv N'$: then by definition of the predicate $\Downarrow$:

$$\frac{unit\ W \Downarrow_0 W \qquad V'[W] \Downarrow_p W'}{unit\ W \star V' \Downarrow_{p+1} W'}$$

$M \equiv N' \star \lambda x.unit\ x \longrightarrow N'$: then we have

$$\frac{N' \Downarrow_p W' \qquad unit\ W' \Downarrow_0 W'}{N' \star \lambda x.unit\ x \Downarrow_{p+1} W'}$$

$M \equiv (L \star \lambda x.M') \star \lambda y.P \longrightarrow L \star \lambda x.(M' \star \lambda y.P) \equiv N'$, where $x \notin FV(P)$. Then $N' \Downarrow W'$ implies that, for some values U, W''

$$\frac{L \Downarrow U \qquad \dfrac{M'[U/x] \Downarrow W'' \qquad P[W''/y] \Downarrow W'}{M'[U/x] \star \lambda y.P \Downarrow W'}}{L \star \lambda x.(M' \star \lambda y.P) \Downarrow W'}$$

where $M'[U/x] \star \lambda y.P \equiv (M' \star \lambda y.P)[U/x]$ since $x \notin FV(P)$. This can be rearranged into

$$\dfrac{\dfrac{L \Downarrow U \qquad M'[U/x] \Downarrow W''}{(L \star \lambda x.M') \Downarrow W'' \qquad\qquad P[W''/y] \Downarrow W'}}{(L \star \lambda x.M') \star \lambda y.P \Downarrow W'}$$

In the remaining cases $M \equiv \mathcal{C}\langle M' \rangle \longrightarrow \mathcal{C}\langle N'' \rangle \equiv N'$ because $M' \longrightarrow N''$.

- $M \equiv M' \star U \longrightarrow N'' \star U \equiv N'$. By principal induction $N' \Downarrow W'$ and $W' \xrightarrow{*} V$. Then $\exists \bar{W}.\, N'' \Downarrow \bar{W}$ and $U[\bar{W}] \Downarrow W'$, so that $M' \Downarrow \bar{V}$ for some $\bar{V}$ s.t. $unit\, \bar{V} \xrightarrow{*} unit\, \bar{W}$ by secondary induction. This implies $unit\, U[\bar{V}] \xrightarrow{*} unit\, U[\bar{W}]$. From $U[\bar{W}] \Downarrow W'$ by secondary induction we obtain that $U[\bar{V}] \Downarrow \bar{U}$ for some $\bar{U}$ s.t. $unit\, \bar{U} \xrightarrow{*} unit\, W'$. Then we conclude that $M' \star U \Downarrow \bar{U}$ and $unit\, \bar{U} \xrightarrow{*} unit\, V$.

- $M \equiv L \star \lambda x.M' \longrightarrow L \star \lambda x.N'' \equiv N'$. By principal induction $N' \Downarrow W'$ and $W' \xrightarrow{*} V$, so that by definition of $\Downarrow$ there exists U such that $L \Downarrow U$ and $N''[U/x] \Downarrow W'$.
  From $M' \longrightarrow N''$ it follows that $M'[U/x] \longrightarrow N''[U/x]$, where $[U/x]$ is a closing substitution of M', N'' since M, N' and U are closed, and hence $FV(M') \cup FV(N'') \subseteq \{x\}$. Therefore by secondary induction $M'[U/x] \Downarrow \bar{U}$ for some $\bar{U}$ s.t. $unit\ \bar{U} \xrightarrow{*} unit\ W'$. In conclusion, $(L \star \lambda x.M') \Downarrow \bar{U}$ and $unit\, \bar{U} \xrightarrow{*} unit\, V$.

□

Theorem 8.4. $M \Downarrow \;\Leftrightarrow\; \exists V.\, M \xrightarrow{*} unit\, V$

Proof. Immediate consequence of Lemma 8.2 and Lemma 8.3. □

In view of Theorem 8.4, the predicate $M \Downarrow$ is non trivial. Indeed consider the closed term:

$$\Omega_{\mathsf{C}} \equiv unit\, (\lambda x.unit\, x \star x) \star (\lambda x.unit\, x \star x)$$

that is a translation of the well known term $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ from ordinary λ-calculus. Then the only reduction out of Ω_{C} is

$$\Omega_{\mathsf{C}} \longrightarrow (unit\, x \star x)[\lambda x.unit\, x \star x/x] \equiv \Omega_{\mathsf{C}} \qquad \text{by } \beta_c$$

which is not of the shape *unit* V for any $V \in \mathit{Val}$, hence $\Omega_{\mathsf{C}} \not\Downarrow$. The main purpose of this section is to show that typing in our system characterizes convergent terms. We say that $\tau \in \mathit{ComType}$ is *non trivial* if $\tau \neq_{\mathsf{C}} \omega_{\mathsf{C}}$. Then we want to show:

Theorem 8.5 (Convergence characterization)**.** *For all $M \in \mathit{Com}^0$ we have:*

$$M \Downarrow \quad \Leftrightarrow \quad \exists \, \textit{non trivial } \tau. \vdash M : \tau.$$

Towards the proof, and following the pattern of Tait's computability method, we introduce some auxiliary notions.

Definition 8.6. *Let $\mathcal{I} : \mathit{TypeVar} \to \mathscr{P}\,\mathit{Val}^0$ be a map; then define $|\delta|_{\mathcal{I}} \subseteq \mathit{Val}^0$ and $|\tau|_{\mathcal{I}} \subseteq \mathit{Com}^0$ by induction as follows:*

i) $|\alpha|_{\mathcal{I}} = \mathcal{I}(\alpha)$

ii) $|\delta \to \tau|_{\mathcal{I}} = \{V \in \mathit{Val}^0 \mid \forall M \in |T\delta|_{\mathcal{I}}.\ M \star V \in |\tau|_{\mathcal{I}}\}$

iii) $|T\delta|_{\mathcal{I}} = \{M \in \mathit{Com}^0 \mid \exists V \in |\delta|_{\mathcal{I}}.\ M \Downarrow V\}$

iv) $|\omega_{\mathsf{V}}|_{\mathcal{I}} = \mathit{Val}^0$ *and* $|\omega_{\mathsf{C}}|_{\mathcal{I}} = \mathit{Com}^0$

v) $|\delta \wedge \delta'|_{\mathcal{I}} = |\delta|_{\mathcal{I}} \cap |\delta'|_{\mathcal{I}}$ *and* $|\tau \wedge \tau'|_{\mathcal{I}} = |\tau|_{\mathcal{I}} \cap |\tau'|_{\mathcal{I}}$.

Lemma 8.7. *Let $\mathcal{I}$ be arbitrary. Then:*

i) $\delta \leq_{\mathsf{V}} \delta' \Rightarrow |\delta|_{\mathcal{I}} \subseteq |\delta'|_{\mathcal{I}}$

ii) $\tau \leq_{\mathsf{C}} \tau' \Rightarrow |\tau|_{\mathcal{I}} \subseteq |\tau'|_{\mathcal{I}}$

Proof. By checking axioms and rules in Definition 6.3. The only non trivial cases concern the arrow and T-types.

Let $V \in |(\delta \to \tau_1) \wedge (\delta \to \tau_2)|_{\mathcal{I}} = |\delta \to \tau_1|_{\mathcal{I}} \cap |\delta \to \tau_2|_{\mathcal{I}}$, then for all $M \in |T\delta|_{\mathcal{I}}$ we have $M \star V \in |\tau_i|_{\mathcal{I}}$ for both $i = 1, 2$; hence $M \star V \in |\tau_1|_{\mathcal{I}} \cap |\tau_2|_{\mathcal{I}} = |\tau_1 \wedge \tau_2|_{\mathcal{I}}$.

Suppose that $\delta_1 \leq_{\mathsf{V}} \delta_2$ and let $M \in |T\delta_1|_{\mathcal{I}}$; then there exists $V \in |\delta_1|_{\mathcal{I}}$ such that $M \Downarrow V$. By induction $|\delta_1|_{\mathcal{I}} \subseteq |\delta_2|_{\mathcal{I}}$ so that immediately we have $M \in |T\delta_2|_{\mathcal{I}}$.

Let $M \in |T\delta_1 \wedge T\delta_2|_{\mathcal{I}} = |T\delta_1|_{\mathcal{I}} \cap |T\delta_2|_{\mathcal{I}}$. Then there exists $V_1 \in |\delta_1|_{\mathcal{I}}$ and $V_2 \in |\delta_2|_{\mathcal{I}}$ such that $M \Downarrow V_1$ and $M \Downarrow V_2$. By Lemma 8.2 we have $M \xrightarrow{*} \mathit{unit}\, V_i$ for both $i = 1, 2$ and these terms are in normal form; hence $V_1 \equiv V_2$ by Theorem 5.12. It follows that there exists a unique $V \in |\delta_1|_{\mathcal{I}} \cap |\delta_2|_{\mathcal{I}} = |\delta_1 \wedge \delta_2|_{\mathcal{I}}$ such that $M \xrightarrow{*} \mathit{unit}\, V$, hence $M \in |T(\delta_1 \wedge \delta_2)|_{\mathcal{I}}$.

Suppose that $\delta_2 \leq_{\mathsf{V}} \delta_1$ and $\tau_1 \leq_{\mathsf{C}} \tau_2$. Let $V \in |\delta_1 \to \tau_1|_{\mathcal{I}}$ and $M \in |T\delta_2|_{\mathcal{I}}$; by the above $M \in |T\delta_1|_{\mathcal{I}}$ so that $M \star V \in |\tau_1|_{\mathcal{I}}$. By induction $|\tau_1|_{\mathcal{I}} \subseteq |\tau_2|_{\mathcal{I}}$ hence $M \star V \in |\tau_2|_{\mathcal{I}}$ so that $V \in |\delta_2 \to \tau_2|_{\mathcal{I}}$ by the choice of M.

□

Corollary 8.8. *A type $\tau \in \mathit{ComType}$ is non trivial if and only if $\tau \leq_{\mathsf{C}} T\omega_{\mathsf{V}}$.*

Proof. By contradiction, if $T\omega_{\mathsf{V}} =_{\mathsf{C}} \omega_{\mathsf{C}}$ then $|T\omega_{\mathsf{V}}|_{\mathcal{I}} = |\omega_{\mathsf{C}}|_{\mathcal{I}}$ for any $\mathcal{I}$ by ii) of Lem. 8.7. But $|T\omega_{\mathsf{V}}|_{\mathcal{I}} = \{M \in \mathit{Com}^0 \mid M\Downarrow\} \neq \mathit{Com}^0 = |\omega_{\mathsf{C}}|_{\mathcal{I}}$ since $\Omega_{\mathsf{C}} \not\Downarrow$. The remaining part of the thesis now follows from Lem. 6.5.

□

We are now in place to show the only if part of Theorem 8.5.

Lemma 8.9. $M\Downarrow \;\Rightarrow\; \exists\, \textit{non trivial}\ \tau.\ \vdash M : \tau$

Proof. If $M \Downarrow$ then $M \xrightarrow{*} \mathit{unit}\, V$ for some $V \in \mathit{Val}^0$ by Lemma 8.2; now $\vdash V : \omega_{\mathsf{V}}$ so that $\vdash \mathit{unit}\, V : T\omega_{\mathsf{V}}$ by rule (unit I); it follows that $\vdash M : T\omega_{\mathsf{V}}$ by Theorem 7.6, where $T\omega_{\mathsf{V}}$ is non trivial by Cor. 8.8. □

We say that a subset $X \subseteq \mathit{Com}^0$ is *saturated* if for all $M \in \mathit{Com}^0$, $M \longrightarrow N$ and $N \in X$ imply $M \in X$.

Lemma 8.10. *For all $\tau \in \mathit{ComType}$ and $\mathcal{I}$ the set $|\tau|_{\mathcal{I}}$ is saturated.*

Proof. By induction over τ. The case $\tau \equiv \omega_{\mathsf{C}}$ is trivial; the case $\tau \equiv \tau_1 \wedge \tau_2$ is immediate by induction. Let $\tau \equiv T\delta$: then by hypothesis there exists $V \in |\delta|_{\mathcal{I}}$ such that $N\Downarrow V$. By Lem. 8.2 we have that $N \xrightarrow{*} \mathit{unit}\, V$ so that $M \xrightarrow{*} \mathit{unit}\, V$ and we conclude by Lem. 8.3. □

Lemma 8.11. *Let $\Gamma \vdash M : \tau$ where $\Gamma = \{x_1 : \delta_1, \ldots, x_k : \delta_k\}$ and $M \in \mathit{Com}$. For any $V_1, \ldots, V_k \in \mathit{Val}^0$ and $\mathcal{I}$, if $V_i \in |\delta_i|_{\mathcal{I}}$ for all $i = 1, \ldots, k$ then $M[V_1/x_1] \cdots [V_k/x_k] \in |\tau|_{\mathcal{I}}$.*

Proof. We strength the thesis by adding that if $\Gamma \vdash W : \delta$ for $W \in \mathit{Val}$, then $W[V_1/x_1] \cdots [V_k/x_k] \in |\delta|_{\mathcal{I}}$ under the same hypotheses. Then we reason by simultaneous induction over the derivations of $\Gamma \vdash M : \tau$ and $\Gamma \vdash W : \delta$. The cases of (Ax) and (ω) are straightforward; cases (unit I) and $(\wedge I)$ are immediate by induction; case $(\leq)$ follows by induction and Lemma 8.7. Let us abbreviate $M[\vec{V}/\vec{x}] \equiv M[V_1/x_1] \cdots [V_k/x_k]$ and similarly for $W[\vec{V}/\vec{x}]$.

Case ($\to$ I): then the derivation ends by:

$$\frac{\Gamma, y : \delta' \vdash M' : \tau'}{\Gamma \vdash \lambda y.M' : \delta' \to \tau'} \ (\to \mathrm{I})$$

where $W \equiv \lambda y.M'$ and $\delta \equiv \delta' \to \tau'$. Let $M'' \equiv M'[\vec{V}/\vec{x}]$ and assume that $y \notin \vec{x}$; to prove that $(\lambda y.M')[\vec{V}/\vec{x}] \equiv \lambda y.M'' \in |\delta' \to \tau'|_{\mathcal{I}}$ we have to show that $N \star \lambda y.M'' \in |\tau'|_{\mathcal{I}}$ for all $N \in |T\delta'|_{\mathcal{I}}$.

Now if $N \in |T\delta'|_{\mathcal{I}}$ then there exists $V' \in |\delta'|_{\mathcal{I}}$ such that $N \Downarrow V'$. This implies that the hypothesis that $V_i \in |\delta_i|_{\mathcal{I}}$ for all $x_i : \delta_i \in \Gamma$ now holds for the larger basis $\Gamma, y : \delta'$ so that by induction we have $M''[V'/y] \in |\tau'|_I$. But

$$N \star \lambda y.M'' \xrightarrow{*} (\mathit{unit}\, V') \star \lambda y.M'' \longrightarrow M''[V'/y]$$

and the thesis follows since $|\tau'|_{\mathcal{I}}$ is saturated by Lemma 8.10.

Case ($\to$ E): then the derivation ends by:

$$\frac{\Gamma \vdash M' : T\delta \quad \Gamma \vdash W' : \delta \to \tau}{\Gamma \vdash M' \star W' : \tau}$$

where $M \equiv M' \star W'$. Let $M'' \equiv M'[\vec{V}/\vec{x}]$ and $W'' \equiv W'[\vec{V}/\vec{x}]$, so that $(M' \star W')[\vec{V}/\vec{x}] \equiv M'' \star W''$. By induction $M'' \in |T\delta|_{\mathcal{I}}$ and $W'' \in |\delta \to \tau|_{\mathcal{I}}$ and the thesis follows by definition of the set $|\delta \to \tau|_{\mathcal{I}}$.

□

We can now finish the proof of Theorem 8.5.

Proof. By Lem. 8.9 it remains to show that if $\vdash M : \tau$ for some non trivial τ then $M \Downarrow$. Since $M \in \mathit{Com}^0$ and the basis is empty, the hypothesis of Lem. 8.11 are vacuously true, so that we have $M \in |\tau|_{\mathcal{I}}$ for all $\mathcal{I}$. On the other hand, by Cor. 8.8, we know that $\tau \leq_{\mathsf{C}} T\omega_{\mathsf{V}}$ since τ is non trivial. By Lem. 8.7 it follows that $M \in |\tau|_{\mathcal{I}} \subseteq |T\omega_{\mathsf{V}}|_{\mathcal{I}} = \{N \in \mathit{Com}^0 \mid N \Downarrow\}$ and we conclude. □

9. Related work and further developments

Since Moggi's seminal papers [1, 2], a substantial body of research has been carried out about the computational λ-calculus and usage of the concept of monad, both in theory and in practice of functional programming languages. Here, because of the large bibliography on the subject, we shall limit ourself, referring just to the closest related works.

The computational λ-calculus has been constructed as the theory of a categorical model; similarly in section 2, we base the definition of the λ_c^u-calculus on strong monads over concrete ccc's possessing a call-by-value reflexive object. As remarked, models having unit and bind as primitive operators are rather different than Moggi's λ_c-models: this is discussed in [19], where in particular the concept of a $\mathcal{C}$-monad seems the appropriate generalization of our functional monad, which is based on a self enriched, concrete ccc.

The calculus λ_c^u in section 3 is similar to Wadler's one in [4], but it includes terms like $unit\ x \star x$, the minimal version of self-application, that obviously has no typed correspondent. Moggi's type free calculus in [1] §6, called λ_c (a name we have been using here for Moggi's typed calculus) is clearly the ancestor of λ_c^u, but the usage of unit and bind in place of the *let*-constructor is not a little change. On the one hand we can define reduction by orienting the three monad laws: this is simpler than having six rules plus η as in Moggi's case. On the other hand, relating the two calculi is not straightforward. Indeed the untyped λ_c includes application of arbitrary terms and admits terms like $let\ x = M\ in\ N$ for non value N; also sorts are not preserved by reduction, and a non-value may reduce to a value, as in call-by-value λ-calculus: however, if this is not disturbing in case of the latter, it produces a semantic mismatch when monads are involved, since D and TD are different domains in general.

In [16] we define a translation $\ulcorner\cdot\urcorner : \lambda_c^u \to \lambda_c$, where $\ulcorner M \star V \urcorner$ is equal to $(let\ x = \ulcorner M \urcorner\ in\ \ulcorner V \urcorner x)$; this preserves conversion, but not reduction. In the opposite direction, we have a translation $\llcorner\cdot\lrcorner : \lambda_c \to \lambda_c^u$ that is as $\llcorner n \lrcorner = \llcorner n \lrcorner^{\mathsf{C}}$ and $\llcorner v \lrcorner = unit\, \llcorner v \lrcorner^{\mathsf{V}}$ where in Moggi's terms n is a non value, v is a value and the translations $\llcorner n \lrcorner^{\mathsf{C}}$ and $\llcorner v \lrcorner^{\mathsf{V}}$ into Com and Val respectively are mutually

defined. E.g. in case of the *let*-expressions we have four clauses:

$$\begin{array}{rcl}
\llcorner let\ x = n\ in\ n' \lrcorner^{\mathsf{C}} & = & \llcorner n \lrcorner^{\mathsf{C}} \star \lambda x. \llcorner n' \lrcorner^{\mathsf{C}} \\
\llcorner let\ x = v\ in\ n' \lrcorner^{\mathsf{C}} & = & unit\, \llcorner v \lrcorner^{\mathsf{V}} \star \lambda x. \llcorner n' \lrcorner^{\mathsf{C}} \\
\llcorner let\ x = n\ in\ v \lrcorner^{\mathsf{C}} & = & \llcorner n \lrcorner^{\mathsf{C}} \star \lambda x. unit\, \llcorner v \lrcorner^{\mathsf{V}} \\
\llcorner let\ x = v\ in\ v' \lrcorner^{\mathsf{C}} & = & unit\, \llcorner v \lrcorner^{\mathsf{V}} \star \lambda x. unit\, \llcorner v' \lrcorner^{\mathsf{V}}
\end{array}$$

This translation preserves reduction when η is dropped from reduction in the untyped λ_c; otherwise we must add η_c to $\lambda\mathbf{C}$, hence losing confluence.

By this, both the confluence proof of λ_c in [10] §8.3, where it is called COMP, and the proof by checking critical pair using the tool PolySOL in [12], cannot be used in our case, and we prefer to prove confluence of $\lambda\mathbf{C}$ from scratch, although following a standard pattern: see e.g. [11]. Confluence of COMP is established in [10] via a translation from a call-by-need linear λ-calculus, but without η, facing a similar difficulty as we mention here at the end of section 4.

The main inspiration for our intersection type system is [13]. In [16] we study the type interpretation over a λ_c^u-model, which is problematic since it is not inductive. We show there that if the equation (5) is solved in a category of algebraic domains by the inverse limit construction, then such interpretation, which we call monadic, exists, and the type system is sound and complete w.r.t. monadic type interpretations.

Intersection types have been used in [30] in a λ-calculus with side effects and reference types. In their work a problem appears, since left distributivity of the arrow over intersection (a rule in [13], that is an axiom of the theory Th_{V} in Definition 6.3 above) is unsound. This is remedied by restricting intersection introduction to values. However, Davies and Pfenning's work is not concerned with monads, so that value and non-value terms and types are of the same sorts. On the contrary, by working in a system like in Definition 6.6, these types are distinct: we conjecture that, if actual definitions of unit and bind for the state monad are consistent with their typings, a type system can be constructed that is an instance of ours, such that it is sound without imposing any ad hoc constraint.

The convergence relation in section 8 is the adaptation of a similar concept introduced in [14, 15] for the lazy λ-calculus, which represents the only observable property in the definition of the applicative bisimulation. It shares some similarity with the convergence relation considered in [8], where Abramsky's idea is extended to a computational λ-calculus very similar to λ_c^u. How-

ever, differently than in Abramsky's work and the relation in Definition 8.1, the authors define their predicate as a relation among syntax and semantics, co-inductively defining the interpretation of terms.

Theorem 8.5, characterizing convergent terms by non trivial typings, is evidence of the expressive power of our system. However, since convergence is undecidable, non-trivial typability in the system is also undecidable. If the system should be useful in practice, say as a method for abstract interpretation and static analysis or program synthesis, then restricted subsystems should be considered, like bounded intersection type systems recently proposed in [31, 32, 33].

Concerning future developments, we see at least three lines of research. The type system we have presented is about a generic monad T: what about typing calculi with specific monads, like partiality, exceptions, state, or input-output? The question itself of what means to instantiate the λ_c^u-calculus and the type assignment system to a particular one, knowing of sufficient conditions guaranteeing that good properties studied here are inherited, is both of theoretical and practical interest.

The λ_c^u-calculus is pure, namely without constants. To formalize data types we need algebraic terms and suitable typing rules; in the framework of intersection type systems, types provide a logical semantics to terms, which is the consequence of type invariance under conversion, and, at the same time, can be seen as a denotational semantics in the category of algebraic domains: see the filter model construction in [13] and Abramsky's domain logic theory [34]. A natural question is then what kind of algebraic and co-algebraic specifications and principles are sound in the logical semantics, when induced by an intersection type system with monads.

The computational λ-calculus has been proposed as a foundation for the static analysis of effectful calculi and programming languages. In [5], extended version of the previously published [35], this is compared to Lucassen and Gifford [36] and Talpin and Jouvelot [37] type and effect discipline (see [38] chap. 5 for an exposition, and the relation to other static analysis techniques). The same topic has been treated by Benton and others in [6] and [7]. While it is known that intersection types and abstract interpretation are related, see [39] and [40], we don't know of any research work relating intersection types to effect systems. Now that we have introduced intersection types for the computational λ-calculi, we have the right theoretical framework to investigate this topic.

10. Conclusion

We have studied how to transfer to the computational λ-calculus, some basic properties and theoretical results of the ordinary, untyped λ-calculus. This leads to a simplified syntax and axiomatization of reduction, together with a Curry style, simple type assignment system, expressive enough to characterize convergent terms by their typings.

Acknowledgments

The authors have been partially supported by COST Action: EUTypes CA15123; and local funds of the University of Turin: Ricerca locale Linea A (BERS-RILO-17-03) - Fondazioni logiche della computazione, Ricerca locale Linea A (PAOL-RILO-18-01) - Fondazioni logiche della computazione, Ricerca locale 2019 Linea A (DE-U-RILO-19-01) - Logica della computazione.

Authors thank anonymous referees for their useful remarks and suggestions.

References

[1] E. Moggi, Computational lambda-calculus and monads, in: Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), IEEE Computer Society, 1989, pp. 14–23. doi:10.1109/LICS.1989.39155.

[2] E. Moggi, Notions of computation and monads, Inf. Comput. 93 (1991) 55–92. doi:10.1016/0890-5401(91)90052-4.

[3] A. Filinski, Representing monads, in: Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press, 1994, pp. 446–457. doi:10.1145/174675.178047.

[4] P. Wadler, Monads for functional programming, in: Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques, volume 925 of *Lecture Notes in Computer Science*, Springer, 1995, pp. 24–52. doi:10.1007/3-540-59451-5_2.

[5] P. Wadler, P. Thiemann, The marriage of effects and monads, ACM Trans. Comput. Log. 4 (2003) 1–32. doi:10.1145/601775.601776.

[6] N. Benton, J. Hughes, E. Moggi, Monads and effects, in: Applied Semantics, International Summer School, APPSEM 2000, volume 2395 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 42–122. doi:10.1007/3-540-45699-6_2.

[7] N. Benton, A. Kennedy, M. Hofmann, L. Beringer, Reading, writing and relations, in: Programming Languages and Systems, 4th Asian Symposium, APLAS 2006, Sydney, Australia, November 8-10, 2006, Proceedings, volume 4279 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 114–130. doi:10.1007/11924661_7.

[8] U. Dal Lago, F. Gavazzo, P. B. Levy, Effectful Applicative Bisimilarity: Monads, Relators, and Howe's Method, in: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, IEEE Computer Society, 2017, pp. 1–12. doi:10.1109/LICS.2017.8005117.

[9] D. Scott, Relating theories of the λ-calculus, in: R. J. Hindley, J. P. Seldin (Eds.), To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism, Academic Press, 1980, pp. 403–450.

[10] J. Maraist, M. Odersky, D. N. Turner, P. Wadler, Call-by-name, call-by-value, call-by-need and the linear lambda calculus, Theor. Comput. Sci. 228 (1999) 175–210. doi:10.1016/S0304-3975(98)00358-2.

[11] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, P. Wadler, The call-by-need lambda calculus, in: Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1995, ACM Press, 1995, pp. 233–246. doi:10.1145/199448.199507.

[12] M. Hamana, Polymorphic rewrite rules: Confluence, type inference, and instance validation, in: Functional and Logic Programming - 14th International Symposium, FLOPS 2018, Proceedings, volume 10818 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 99–115. doi:10.1007/978-3-319-90686-7_7.

[13] H. P. Barendregt, W. Dekkers, R. Statman, Lambda Calculus with Types, Perspectives in logic, Cambridge University Press, 2013.

[14] S. Abramsky, The lazy lambda calculus, in: Research topics in functional programming, Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 1990, pp. 65–116.

[15] S. Abramsky, C.-H. Ong, Full abstraction in the lazy lambda calculus, Inf. Comput. 105 (1993) 159–267. doi:10.1006/inco.1993.1044.

[16] U. de'Liguoro, R. Treglia, Intersection types for the computational lambda-calculus, CoRR abs/1907.05706 (2019). arXiv:1907.05706.

[17] S. MacLane, Categories for the Working Mathematician, Graduate Texts in Mathematics, 2 ed., Springer, 1997.

[18] P. Wadler, The essence of functional programming, in: Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1992, ACM Press, 1992, pp. 1–14. doi:10.1145/143165.143169.

[19] J. Power, Models for the computational lambda-calculus, Electron. Notes Theor. Comput. Sci. 40 (2000) 288–301. doi:10.1016/S1571-0661(05)80056-8.

[20] A. R. Meyer, What is a model of the lambda calculus?, Inf. Control. 52 (1982) 87–122. doi:10.1016/S0019-9958(82)80087-9.

[21] H. Barendregt, The Lambda Calculus: its Syntax and Semantics, volume 103 of *Studies in logic and the foundations of mathematics*, revised ed., North-Holland, 1985.

[22] G. D. Plotkin, Call-by-name, call-by-value and the lambda-calculus, Theor. Comput. Sci. 1 (1975) 125–159. doi:10.1016/0304-3975(75)90017-1.

[23] M. Takahashi, Parallel reductions in lambda-calculus, Inf. Comput. 118 (1995) 120–127. doi:10.1006/inco.1995.1057.

[24] Terese, Term rewriting systems, volume 55 of *Cambridge tracts in theoretical computer science*, Cambridge University Press, 2003.

[25] F. Baader, T. Nipkow, Term rewriting and all that, Cambridge University Press, 1998.

[26] H. Barendregt, M. Coppo, M. Dezani-Ciancaglini, A filter lambda model and the completeness of type assignment, J. Symb. Log. 48 (1983) 931–940. doi:10.2307/2273659.

[27] P.-A. Melliès, N. Zeilberger, Functors are type refinement systems, in: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, ACM, 2015, pp. 3–16. doi:10.1145/2676726.2676970.

[28] M. Coppo, M. Dezani-Ciancaglini, F. Honsell, G. Longo, Extended type structures and filter lambda models, in: G. Lolli, G. Longo, A. Marcja (Eds.), Logic Colloquium 82, North-Holland, Amsterdam, the Netherlands, 1984, pp. 241–262.

[29] G. D. Plotkin, A structural approach to operational semantics, J. Log. Algebraic Methods Program. 60-61 (2004) 17–139.

[30] R. Davies, F. Pfenning, Intersection types and computational effects, in: Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), ACM, 2000, pp. 198–208. doi:10.1145/351240.351259.

[31] B. Düdder, M. Martens, J. Rehof, P. Urzyczyn, Bounded combinatory logic, in: Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, volume 16 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 243–258. doi:10.4230/LIPIcs.CSL.2012.243.

[32] A. Dudenhefner, J. Rehof, Intersection type calculi of bounded dimension, in: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, ACM, 2017, pp. 653–665. URL: http://dl.acm.org/citation.cfm?id=3009862.

[33] A. Dudenhefner, J. Rehof, Typability in bounded dimension, in: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, IEEE Computer Society, 2017, pp. 1–12. doi:10.1109/LICS.2017.8005127.

[34] S. Abramsky, Domain theory in logical form, Ann. Pure Appl. Log. 51 (1991) 1–77. doi:10.1016/0168-0072(91)90065-T.

[35] P. Wadler, The marriage of effects and monads, in: Proceedings of the third ACM SIGPLAN International Conference on Functional Programming (ICFP '98, ACM, 1998, pp. 63–74. doi: 10.1145/289423.289429.

[36] J. M. Lucassen, D. K. Gifford, Polymorphic effect systems, in: Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, California, USA, January 10-13, 1988, ACM Press, 1988, pp. 47–57. doi: 10.1145/73560.73564.

[37] J. Talpin, P. Jouvelot, The type and effect discipline, Inf. Comput. 111 (1994) 245–296. doi: 10.1006/inco.1994.1046.

[38] F. Nielson, H. R. Nielson, C. Hankin, Principles of program analysis, Springer, 1999. doi: 10.1007/978-3-662-03811-6.

[39] T. Jensen, Conjunctive Type Systems and Abstract Interpretation of Higher-order Functional Programs, Journal of Logic and Computation 5 (1995) 397–421. doi: 10.1093/logcom/5.4.397.

[40] M. Coppo, A. Ferrari, Type inference, abstract interpretation and strictness analysis, Theor. Comput. Sci. 121 (1993) 113–143. doi: 10.1016/0304-3975(93)90086-9.